

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

v2.0.1

Sašo Živanović*

February 20, 2016

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

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

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

Contents

1	Identification	2
2	Package options	2
3	Patches	3
4	Utilities	3
4.1	Sorting	9
5	The bracket representation parser	13
5.1	The user interface macros	13
5.2	Parsing	14
5.3	The tree-structure interface	18
6	Nodes	19
6.1	Option setting and retrieval	19
6.2	Tree structure	22
6.3	Node options	30
6.3.1	Option-declaration mechanism	30
6.3.2	Registers	37
6.3.3	Declaring options	43
6.3.4	Option propagation	49
6.4	Aggregate functions	51
6.4.1	<code>pgfmath</code> extensions	52
6.5	Nodewalk	55
6.6	Dynamic tree	73
7	Stages	77
7.1	Typesetting nodes	80
7.2	Packing	82
7.2.1	Tiers	92
7.2.2	Node boundary	96
7.3	Compute absolute positions	101
7.4	Drawing the tree	101

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

8	Geometry	103
8.1	Projections	103
8.2	Break path	106
8.3	Get tight edge of path	108
8.4	Get rectangle/band edge	114
8.5	Distance between paths	115
8.6	Utilities	118
9	The outer UI	119
9.1	Externalization	119
9.2	The <code>forest</code> environment	121
9.3	Standard node	124
9.4	ls coordinate system	125
9.5	Relative node names in <code>TikZ</code>	126
9.6	Anchors	127
10	Compatibility with previous versions	132

1 Identification

```

1 \ProvidesPackage{forest}[2016/02/20 v2.0.1 Drawing (linguistic) trees]
2
3 \RequirePackage{tikz}[2013/12/13]
4 \usetikzlibrary{shapes}
5 \usetikzlibrary{fit}
6 \usetikzlibrary{calc}
7 \usepgflibrary{intersections}
8
9 \RequirePackage{pgfopts}
10 \RequirePackage{etoolbox}[2010/08/21]
11 \RequirePackage{elocalloc}% for \locbox
12 \RequirePackage{environ}
13 \RequirePackage{xparse}
14
15 % \usepackage[trace]{trace-pgfkeys}
16
17 /forest is the root of the key hierarchy.
17 \pgfkeys{/forest/.is family}
18 \def\forestset#1{\pgfqkeys{/forest}{#1}}
```

2 Package options

```

19 \newif\ifforest@external@
20 \newif\ifforesttikzcshack
21 \newif\ifforest@install@keys@to@tikz@path@
22 \newif\ifforestdebug
23 \def\forest@compat{}
24 \forestset{package@options/.cd,
25   external/.is if=forest@external@,
26   tikzcshack/.is if=foresttikzcshack,
27   tikzinstallkeys/.is if=forest@install@keys@to@tikz@path@,
28   debug/.is if=forestdebug,
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 \foresttikzcshacktrue
40 \def\forest@loadlibrarieslater{}
41 \AtEndOfPackage{\forest@loadlibrarieslater}
42 \NewDocumentCommand\useforestlibrary{s m}{%
43   \forcsvlist\useforestlibrary@{#2}%
44   \IfBooleanT{#1}{\forestapplylibrarydefaults{#2}}%
45 }
46 \def\useforestlibrary@#1{\RequirePackage{forest-lib-#1}}
47 \def\forestapplylibrarydefaults#1{\forcsvlist\forestapplylibrarydefaults@{#1}}
48 \def\forestapplylibrarydefaults@#1{\forestset{libraries/#1/defaults/.try}}
49 \NewDocumentCommand\ProvidesForestLibrary{m O{}}{\ProvidesPackage{forest-lib-#1}[#2]}
50 \ProcessPgfPackageOptions{/forest/package@options}

```

3 Patches

This macro implements a fairly safe patching mechanism: the code is only patched if the original hasn't changed. If it did change, a warning message is printed. (This produces a spurious warning when the new version of the code fixes something else too, but what the heck.)

```

51 \def\forest@patch#1#2#3#4#5{%
52   % #1 = cs to be patched
53   % #2 = purpose of the patch
54   % #3 = macro arguments
55   % #4 = original code
56   % #5 = patched code
57   \csdef{forest@original@#1}#3{#4}%
58   \csdef{forest@patched@#1}#3{#5}%
59   \ifcsequal{#1}{forest@original@#1}{%
60     \csletcs{#1}{forest@patched@#1}%
61   }{%
62     \ifcsequal{#1}{forest@patched@#1}{% all is good, the patch is in!
63     }{%
64       \PackageWarning{forest}{Failed patching '\expandafter\string\csname #1\endcsname'. Purpose of the patch
65     }%
66   }%
67 }

```

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

```

68 \forest@patch{pgfgettransform}{fix a leaking space}{#1}{%
69   \edef#1{{\pgf@pt@aa}{\pgf@pt@ab}{\pgf@pt@ba}{\pgf@pt@bb}{\the\pgf@pt@x}{\the\pgf@pt@y}}
70 }{%
71   \edef#1{{\pgf@pt@aa}{\pgf@pt@ab}{\pgf@pt@ba}{\pgf@pt@bb}{\the\pgf@pt@x}{\the\pgf@pt@y}}%
72 }

```

4 Utilities

Escaping \ifs.

```

73 \long\def\@escapeif#1#2\fi{\fi#1}
74 \long\def\@escapeifif#1#2\fi#3\fi{\fi\fi#1}
75 \long\def\@escapeififif#1#2\fi#3\fi#4\fi{\fi\fi\fi#1}

```

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

```

76 \def\newloop#1{%
77   \count@=\escapechar
78   \escapechar=-1
79   \expandafter\newloop@parse@loopname\string#1\newloop@end
80   \escapechar=\count@
81 }%

```

```

82 {\lccode'7='1 \lccode'8='o \lccode'9='p
83 \lowercase{\gdef\newloop@parse@loopname#17889#2\newloop@end{%
84   \edef\newloop@marshal{%
85     \noexpand\csdef{#1loop#2}####1\expandafter\noexpand\csname #1repeat#2\endcsname{%
86       \noexpand\csdef{#1iterate#2}{####1\relax\noexpand\expandafter\expandafter\noexpand\csname#1iterate#
87       \expandafter\noexpand\csname#1iterate#2\endcsname
88       \let\expandafter\noexpand\csname#1iterate#2\endcsname\relax
89     }%
90   }%
91   \newloop@marshal
92 }%
93 }%
94 }%

```

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.

```

95 \def\newsafeloop#1{%
96   \csdef{safeloop@#1}##1\saferepeat{%
97     \edef\safeloop@marshal{%
98       \noexpand\csdef{safeiterate@#1}{%
99         \unexpanded{##1}\relax
100         \noexpand\expandafter
101         \expandonce{\csname safeiterate@#1\endcsname}%
102         \noexpand\fi
103       }%
104     }\safeloop@marshal
105     \csuse{safeiterate@#1}%
106     \advance\noexpand\safeloop@depth-1\relax
107     \cslet{safeiterate@#1}\relax
108   }%
109 }%
110 \newcount\safeloop@depth
111 \def\safeloop{%
112   \advance\safeloop@depth1
113   \ifcsdef{safeloop@the\safeloop@depth}{\expandafter\newsafeloop\expandafter{\the\safeloop@depth}}%
114   \csdef{safeloopn@the\safeloop@depth}{0}%
115   \csuse{safeloop@the\safeloop@depth}%
116   \csdef{safeloopn@the\safeloop@depth}{\number\numexpr\csuse{safeloopn@the\safeloop@depth}+1}%
117 }
118 \let\saferepeat\fi
119 \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.

```

120 \def\newsafeRKloop#1{%
121   \csdef{safeRKloop@#1}##1\safeRKrepeat{%
122     \edef\safeRKloop@marshal{%
123       \noexpand\csdef{safeRKiterate@#1}{%
124         \unexpanded{##1}\relax
125         \noexpand\expandafter
126         \expandonce{\csname safeRKiterate@#1\endcsname}%
127         \noexpand\fi
128       }%
129     }\safeRKloop@marshal
130     \csuse{safeRKiterate@#1}%
131     \advance\noexpand\safeRKloop@depth-1\relax
132     \cslet{safeRKiterate@#1}\relax
133   }%
134   \expandafter\newif\csname ifsafeRKbreak@the\safeloop@depth\endcsname
135 }%

```

```

136 \newcount\safeRKloop@depth
137 \def\safeRKloop{%
138   \advance\safeRKloop@depth1
139   \ifcsdef{safeRKloop@\the\safeRKloop@depth}{\expandafter\newsafeRKloop\expandafter{\the\safeRKloop@depth}}%
140   \csdef{safeRKloopn@\the\safeRKloop@depth}{0}%
141   \csuse{safeRKbreak@\the\safeRKloop@depth false}%
142   \csuse{safeRKloop@\the\safeRKloop@depth}%
143   \csdef{safeRKloopn@\the\safeRKloop@depth}{\number\numexpr\csuse{safeRKloopn@\the\safeRKloop@depth}+1}%
144 }
145 \let\safeRKrepeat\fi
146 \def\safeRKloopn{\csuse{safeRKloopn@\the\safeRKloop@depth}}%
  Additional loops (for embedding).
147 \newloop\forest@loop
  New counters, dimens, ifs.
148 \newdimen\forest@temp@dimen
149 \newcount\forest@temp@count
150 \newcount\forest@n
151 \newif\ifforest@temp
152 \newcount\forest@temp@global@count
153 \newtoks\forest@temp@toks
  Appending and prepending to token lists.
154 \def\etotoks#1#2{\edef\pot@temp{#2}\expandafter#1\expandafter{\pot@temp}}
155 \def\apptotoks#1#2{\expandafter#1\expandafter{\the#1#2}}
156 \long\def\lapptotoks#1#2{\expandafter#1\expandafter{\the#1#2}}
157 \def\eapptotoks#1#2{\edef\pot@temp{#2}\expandafter\expandafter\expandafter#1\expandafter\expandafter\expandafter\expandafter
158 \def\pretotoks#1#2{\toks@={#2}\expandafter\expandafter\expandafter#1\expandafter\expandafter\expandafter\expandafter{\exp
159 \def\epretotoks#1#2{\edef\pot@temp{#2}\expandafter\expandafter\expandafter#1\expandafter\expandafter\expandafter\expandafter
160 \def\gapptotoks#1#2{\expandafter\global\expandafter#1\expandafter{\the#1#2}}
161 \def\xapptotoks#1#2{\edef\pot@temp{#2}\expandafter\expandafter\expandafter\global\expandafter\expandafter\expandafter\exp
162 \def\gpretotoks#1#2{\toks@={#2}\expandafter\expandafter\expandafter\global\expandafter\expandafter\expandafter\exp
163 \def\xpretotoks#1#2{\edef\pot@temp{#2}\expandafter\expandafter\expandafter\global\expandafter\expandafter\expandafter\exp
  Expanding number arguments.
164 \def\expandnumberarg#1#2{\expandafter#1\expandafter{\number#2}}
165 \def\expandtwonumberargs#1#2#3{%
166   \expandafter\expandtwonumberargs@\expandafter#1\expandafter{\number#3}{#2}}
167 \def\expandtwonumberargs@#1#2#3{%
168   \expandafter#1\expandafter{\number#3}{#2}}
169 \def\expandthreenumberargs#1#2#3#4{%
170   \expandafter\expandthreenumberargs@\expandafter#1\expandafter{\number#4}{#2}{#3}}
171 \def\expandthreenumberargs@#1#2#3#4{%
172   \expandafter\expandthreenumberargs@@\expandafter#1\expandafter{\number#4}{#2}{#3}}
173 \def\expandthreenumberargs@@#1#2#3#4{%
174   \expandafter#1\expandafter{\number#4}{#2}{#3}}
  A macro converting all non-alphanumerics (and an initial number) in a string to __. #1 = string, #2
  = receiving macro. Used for declaring pgfmath functions.
175 \def\forest@convert@others@to@underscores#1#2{%
176   \def\forest@cotu@result{}}%
177   \forest@cotu@first#1\forest@end
178   \let#2\forest@cotu@result
179 }
180 \def\forest@cotu{%
181   \let\forest@cotu@have@num\forest@cotu@have@alpha
182   \futurelet\forest@cotu@nextchar\forest@cotu@checkforspace
183 }
184 \def\forest@cotu@first{%
185   \let\forest@cotu@have@num\forest@cotu@have@other
186   \futurelet\forest@cotu@nextchar\forest@cotu@checkforspace
187 }

```

```

188 \def\forest@cotu@checkforspace{%
189   \expandafter\ifx\space\forest@cotu@nextchar
190   \let\forest@cotu@next\forest@cotu@havespace
191   \else
192     \let\forest@cotu@next\forest@cotu@nospace
193   \fi
194   \forest@cotu@next
195 }
196 \def\forest@cotu@havespace#1{%
197   \appto\forest@cotu@result{_}%
198   \forest@cotu#1%
199 }
200 \def\forest@cotu@nospace{%
201   \ifx\forest@cotu@nextchar\forest@end
202     \escapeif\@gobble
203   \else
204     \escapeif\forest@cotu@nospaceB
205   \fi
206 }
207 \def\forest@cotu@nospaceB#1{%
208   \ifcat#1a%
209     \let\forest@cotu@next\forest@cotu@have@alpha
210   \else
211     \if!\ifnum9<1#1!\fi
212     \let\forest@cotu@next\forest@cotu@have@num
213   \else
214     \let\forest@cotu@next\forest@cotu@haveother
215   \fi
216   \fi
217   \forest@cotu@next#1%
218 }
219 \def\forest@cotu@have@alpha#1{%
220   \appto\forest@cotu@result{#1}%
221   \forest@cotu
222 }
223 \def\forest@cotu@haveother#1{%
224   \appto\forest@cotu@result{_}%
225   \forest@cotu
226 }

```

Additional list macros.

```

227 \def\forest@listedel#1#2{% #1 = list, #2 = item
228   \edef\forest@marshal{\noexpand\forest@listdel\noexpand#1{#2}}%
229   \forest@marshal
230 }
231 \def\forest@listcsdel#1#2{%
232   \expandafter\forest@listdel\csname #1\endcsname{#2}%
233 }
234 \def\forest@listcsedel#1#2{%
235   \expandafter\forest@listedel\csname #1\endcsname{#2}%
236 }
237 \edef\forest@restorelistsepcatcode{\noexpand\catcode'\the\catcode'\relax}%
238 \catcode'\|=3
239 \gdef\forest@listdel#1#2{%
240   \def\forest@listedel@A##1|#2|##2\forest@END{%
241     \forest@listedel@B##1|##2\forest@END%|
242   }%
243   \def\forest@listedel@B|##1\forest@END{%|
244     \def#1{##1}%
245   }%
246   \expandafter\forest@listedel@A\expandafter|#1\forest@END%|

```

```

247 }
248 \forest@restorelistsepcatcode
    Strip (the first level of) braces from all the tokens in the argument.
249 \def\forest@strip@braces#1{%
250   \forest@strip@braces@A#1\forest@strip@braces@preend\forest@strip@braces@end
251 }
252 \def\forest@strip@braces@A#1#2\forest@strip@braces@end{%
253   #1\ifx\forest@strip@braces@preend#2\else\@escapeif{\forest@strip@braces@A#2\forest@strip@braces@end}\fi
254 }

    Utilities dealing with pgfkeys.
255 \def\forest@copycommandkey#1#2{% copies command of #1 into #2
256   \pgfkeysifdefined{#1/.@cmd}{%
257     \PackageError{forest}{Key #1 is not a command key}{%
258     }%
259     \pgfkeysgetvalue{#1/.@cmd}\forest@temp
260     \pgfkeyslet{#2/.@cmd}\forest@temp
261     \pgfkeysifdefined{#1/.@args}{%
262       \pgfkeysgetvalue{#1/.@args}\forest@temp
263       \pgfkeyslet{#2/.@args}\forest@temp
264     }{}%
265     \pgfkeysifdefined{#1/.@body}{%
266       \pgfkeysgetvalue{#1/.@body}\forest@temp
267       \pgfkeyslet{#2/.@body}\forest@temp
268     }{}%
269     \pgfkeysifdefined{#1/.@@body}{%
270       \pgfkeysgetvalue{#1/.@@body}\forest@temp
271       \pgfkeyslet{#2/.@@body}\forest@temp
272     }{}%
273     \pgfkeysifdefined{#1/.@def}{%
274       \pgfkeysgetvalue{#1/.@def}\forest@temp
275       \pgfkeyslet{#2/.@def}\forest@temp
276     }{}%
277 }
278 \forestset{
279   copy command key/.code 2 args={\forest@copycommandkey{#1}{#2}},
280   autoforward/.code 2 args={\forest@autoforward{#1}{#2={#1={##1}}}{true}},
281   autoforward'/.code 2 args={\forest@autoforward{#1}{#2==#1,#2={#1={##1}}}{true}},
282   Autoforward/.code 2 args={\forest@autoforward{#1}{#2}{true}},
283   autoforward register/.code 2 args={\forest@autoforward{#1}{#2={#1={##1}}}{false}},
284   autoforward register'/.code 2 args={\forest@autoforward{#1}{#2==#1,#2={#1={##1}}}{false}},
285   Autoforward register/.code 2 args={\forest@autoforward{#1}{#2}{false}},
286   copy command key@if it exists/.code 2 args={%
287     \pgfkeysifdefined{#1/.@cmd}{%
288       \forest@copycommandkey{#1}{#2}%
289     }{}%
290   },
291   unautoforward/.style={
292     typeout={unautoforwarding #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     copy command key@if it exists={/forest/autoforwarded #1*}{/forest/#1*},
297     copy command key@if it exists={/forest/autoforwarded #1:}{/forest/#1:},
298     copy command key@if it exists={/forest/autoforwarded #1'}{/forest/#1'},
299     copy command key@if it exists={/forest/autoforwarded #1+'}{/forest/#1+'},
300     copy command key@if it exists={/forest/autoforwarded #1-'}{/forest/#1-'},
301     copy command key@if it exists={/forest/autoforwarded #1*'}{/forest/#1*'},
302     copy command key@if it exists={/forest/autoforwarded #1:'}{/forest/#1:'},
303     copy command key@if it exists={/forest/autoforwarded +#1}{/forest/+ #1},
304   },

```

```

305 /handlers/.undef/.code={\csundef{pgfk@pgfkeyscurrentpath}},
306 undef option/.style={
307   /forest/#1/.undef,
308   /forest/#1/.@cmd/.undef,
309   /forest/#1+/.@cmd/.undef,
310   /forest/#1-/.@cmd/.undef,
311   /forest/#1*/.@cmd/.undef,
312   /forest/#1:/.@cmd/.undef,
313   /forest/#1'/.@cmd/.undef,
314   /forest/#1+'/.@cmd/.undef,
315   /forest/#1-'/.@cmd/.undef,
316   /forest/#1*'/.@cmd/.undef,
317   /forest/#1:'/.@cmd/.undef,
318   /forest/+#1/.@cmd/.undef,
319   /forest/TeX={\patchcmd{\forest@node@init}{\forest@init{#1}}{}{}},
320 },
321 undef register/.style={undef option={#1}},
322 }
323 \def\forest@autoforward#1#2#3{%
324   % #1 = option name
325   % #2 = code of a style taking one arg (new option value),
326   %       which expands to whatever should be done with the new value
327   %       autoforward(') adds to the keylist (arg#2)
328   % #3 = true=option, false=register
329   \forest@autoforward@createforwarder{}{#1}{#2}{#3}%
330   \forest@autoforward@createforwarder{}{#1}{+}{#2}{#3}%
331   \forest@autoforward@createforwarder{}{#1}{-}{#2}{#3}%
332   \forest@autoforward@createforwarder{}{#1}{*}{#2}{#3}%
333   \forest@autoforward@createforwarder{}{#1}{:}{#2}{#3}%
334   \forest@autoforward@createforwarder{}{#1}{'}{#2}{#3}%
335   \forest@autoforward@createforwarder{}{#1}{+'}{#2}{#3}%
336   \forest@autoforward@createforwarder{}{#1}{-'}{#2}{#3}%
337   \forest@autoforward@createforwarder{}{#1}{*'}{#2}{#3}%
338   \forest@autoforward@createforwarder{}{#1}{:'}{#2}{#3}%
339   \forest@autoforward@createforwarder{+}{#1}{#2}{#3}%
340 }
341 \def\forest@autoforward@createforwarder#1#2#3#4#5{%
342   % #1=prefix, #2=option name, #3=suffix, #4=macro code (#2 above), #5=option or register
343   \pgfkeysifdefined{/forest/#1#2#3/.@cmd}{%
344     \forest@copycommandkey{/forest/#1#2#3}{/forest/autoforwarded #1#2#3}%
345     \pgfkeyssetvalue{/forest/autoforwarded #1#2#3/option@name}{#2}%
346     \pgfkeysdef{/forest/#1#2#3}{%
347       \pgfkeysalso{autoforwarded #1#2#3={#1}}%
348       \def\forest@temp@macro###1{#4}%
349       \csname forest@temp#5\endcsname
350       \edef\forest@temp@value{\ifforest@temp\expandafter\forest@v\expandafter{\expandafter\forest@setter@node
351         \expandafter\expandafter\expandafter\expandafter\expandafter\expandafter\expandafter\pgfkeysalso\expandafter
352       }}%
353     }{}%
354 }
355 \def\forest@node@removekeysfromkeylist#1#2{% #1 = keys to remove, #2 = option name
356   \edef\forest@marshal{%
357     \noexpand\forest@removekeysfromkeylist{\unexpanded{#1}}{\forest@v{#2}}\noexpand\forest@temp@toks\forest@
358     \forest@set{#2}{\the\forest@temp@toks}%
359 }
360 \def\forest@removekeysfromkeylist#1#2#3{%
361   % #1 = keys to remove (a keylist: an empty value means remove a key with any value)
362   % #2 = keylist
363   % #3 = toks cs for result
364   \forest@temp@toks{}%
365   \def\forest@novalue{\forest@novalue}%

```



```

366 \pgfqkeys{/forest/remove@key@installer}{#1}%
367 \let\forestnovalue\pgfkeysnovaluertext
368 \pgfqkeys{/forest/remove@key}{#2}%
369 \pgfqkeys{/forest/remove@key@uninstaller}{#1}%
370 #3\forest@temp@toks
371 }
372 \def\forest@remove@key@novalue{\forest@remove@key@novalue}%
373 \forestset{
374   remove@key@installer/.unknown/.code={% #1 = (outer) value
375     \def\forest@temp{#1}%
376     \ifx\forest@temp\pgfkeysnovalue@text
377       \pgfkeysdef{/forest/remove@key/\pgfkeyscurrentname}{}%
378     \else
379       \ifx\forest@temp\forestnovalue
380         \expandafter\forest@remove@key@installer@defwithvalue\expandafter{\pgfkeyscurrentname}{\pgfkeysnovalu
381       \else
382         \expandafter\forest@remove@key@installer@defwithvalue\expandafter{\pgfkeyscurrentname}{#1}%
383       \fi
384     \fi
385   },
386   remove@key/.unknown/.code={% #1 = (inner) value
387     \expandafter\apptotoks\expandafter\forest@temp@toks\expandafter{\pgfkeyscurrentname={#1},}%
388   },
389   remove@key@uninstaller/.unknown/.code={%
390     \pgfkeyslet{/forest/remove@key/\pgfkeyscurrentname/.@cmd}\@undefined},
391 }
392 \def\forest@remove@key@installer@defwithvalue#1#2{% #1=key name, #2 = outer value
393   \pgfkeysdef{/forest/remove@key/#1}{% ##1 = inner value
394     \def\forest@temp@outer{#2}%
395     \def\forest@temp@inner{##1}%
396     \ifx\forest@temp@outer\forest@temp@inner
397     \else
398       \apptotoks\forest@temp@toks{#1={##1},}%
399     \fi
400   }%
401 }

```

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

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

```

402 \def\forest@sort#1#2#3#4#5{%
403   \let\forest@sort@cmp#1\relax
404   \let\forest@sort@let#2\relax
405   \let\forest@sort@direction#3\relax

```

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`.

```

406 \forest@@sort{#4}{#5}%
407 }

```

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.

```

408 \def\forest@quicksort@minarraylength{10000}
409 \def\forest@@sort#1#2{%
410   \ifnum#1<#2\relax\@escapeif{%
411     \forest@sort@m=#2
412     \advance\forest@sort@m -#1
413     \ifnum\forest@sort@m>\forest@quicksort@minarraylength\relax\@escapeif{%
414       \forest@quicksort{#1}{#2}%
415     }\else\@escapeif{%
416       \forest@insertionsort{#1}{#2}%
417     }\fi
418   }\fi
419 }

```

Various counters and macros needed by the sorting algorithms.

```

420 \newcount\forest@sort@m\newcount\forest@sort@k\newcount\forest@sort@p
421 \def\forest@sort@ascending{>}
422 \def\forest@sort@descending{<}
423 \def\forest@sort@cmp{%
424   \PackageError{sort}{You must define forest@sort@cmp function before calling
425     sort}{The macro must take two arguments, indices of the array
426     elements to be compared, and return '=' if the elements are equal
427     and '>'/<' if the first is greater /less than the second element.}%
428 }
429 \def\forest@sort@cmp@gt{\def\forest@sort@cmp@result{>}}
430 \def\forest@sort@cmp@lt{\def\forest@sort@cmp@result{<}}
431 \def\forest@sort@cmp@eq{\def\forest@sort@cmp@result{=}}
432 \def\forest@sort@let{%
433   \PackageError{sort}{You must define forest@sort@let function before calling
434     sort}{The macro must take two arguments, indices of the array:
435     element 2 must be copied onto element 1.}%
436 }

```

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

```

437 \newloop\forest@sort@loop
438 \newloop\forest@sort@loopA
439 \def\forest@quicksort#1#2{%

```

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

```

440   \forest@sort@m=#2
441   \advance\forest@sort@m -#1
442   \ifodd\forest@sort@m\relax\advance\forest@sort@m1 \fi
443   \divide\forest@sort@m 2
444   \advance\forest@sort@m #1

```

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

```

445   \forest@sort@cmp{#1}{#2}%
446   \if\forest@sort@cmp@result=%
447     \forest@sort@p=#1
448   \else
449     \if\forest@sort@cmp@result>%
450       \forest@sort@p=#1\relax
451     \else
452       \forest@sort@p=#2\relax
453     \fi
454   \forest@sort@cmp{\the\forest@sort@p}{\the\forest@sort@m}%
455   \if\forest@sort@cmp@result<%
456     \else
457     \forest@sort@p=\the\forest@sort@m

```

```

458   \fi
459   \fi
    Exchange the pivot and the first element.
460   \forest@sort@xch{#1}{\the\forest@sort@p}%
    Counter \forest@sort@m will hold the final location of the pivot element.
461   \forest@sort@m=#1\relax
    Loop through the list.
462   \forest@sort@k=#1\relax
463   \forest@sort@loop
464   \ifnum\forest@sort@k<#2\relax
465     \advance\forest@sort@k 1
    Compare the pivot and the current element.
466     \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.
467     \ifx\forest@sort@direction\forest@sort@cmp@result
468       \advance\forest@sort@m 1
469       \forest@sort@xch{\the\forest@sort@m}{\the\forest@sort@k}
470     \fi
471   \forest@sort@repeat
    Move the pivot element into its final position.
472   \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.
473   \forest@sort@k=\forest@sort@m
474   \advance\forest@sort@k -1
475   \advance\forest@sort@m 1
476   \edef\forest@sort@marshal{%
477     \noexpand\forest@sort@{#1}{\the\forest@sort@k}%
478     \noexpand\forest@sort@{\the\forest@sort@m}{#2}%
479   }%
480   \forest@sort@marshal
481 }
482 % We defines the item-exchange macro in terms of the (user-provided)
483 % array let macro.
484 %   \begin{macrocode}
485 \def\forest@sort@aux{aux}
486 \def\forest@sort@xch#1#2{%
487   \forest@sort@let{\forest@sort@aux}{#1}%
488   \forest@sort@let{#1}{#2}%
489   \forest@sort@let{#2}{\forest@sort@aux}%
490 }
    Insertion sort.
491 \def\forest@insertionsort#1#2{%
492   \forest@sort@m=#1
493   \edef\forest@insertionsort@low{#1}%
494   \forest@sort@loopA
495   \ifnum\forest@sort@m<#2
496     \advance\forest@sort@m 1
497     \forest@insertionsort@Qbody
498   \forest@sort@repeatA
499 }
500 \newif\ifforest@insertionsort@loop
501 \def\forest@insertionsort@Qbody{%
502   \forest@sort@let{\forest@sort@aux}{\the\forest@sort@m}%

```

```

503 \forest@sort@k\forest@sort@m
504 \advance\forest@sort@k -1
505 \forest@insertionsort@looptrue
506 \forest@sort@loop
507 \ifforest@insertionsort@loop
508   \forest@insertionsort@qbody
509 \forest@sort@repeat
510 \advance\forest@sort@k 1
511 \forest@sort@let{\the\forest@sort@k}{\forest@sort@aux}%
512 }
513 \def\forest@insertionsort@qbody{%
514   \forest@sort@cmp{\the\forest@sort@k}{\forest@sort@aux}%
515   \ifx\forest@sort@direction\forest@sort@cmp@result\relax
516     \forest@sort@p=\forest@sort@k
517     \advance\forest@sort@p 1
518     \forest@sort@let{\the\forest@sort@p}{\the\forest@sort@k}%
519     \advance\forest@sort@k -1
520     \ifnum\forest@sort@k<\forest@insertionsort@low\relax
521       \forest@insertionsort@loopfalse
522     \fi
523   \else
524     \forest@insertionsort@loopfalse
525   \fi
526 }

```

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

Compare numbers.

```

527 \def\forest@sort@cmpnumcs#1#2{%
528   \ifnum\csname#1\endcsname>\csname#2\endcsname\relax
529     \forest@sort@cmp@gt
530   \else
531     \ifnum\csname#1\endcsname<\csname#2\endcsname\relax
532       \forest@sort@cmp@lt
533     \else
534       \forest@sort@cmp@eq
535     \fi
536   \fi
537 }

```

Compare dimensions.

```

538 \def\forest@sort@cmpdimcs#1#2{%
539   \ifdim\csname#1\endcsname>\csname#2\endcsname\relax
540     \forest@sort@cmp@gt
541   \else
542     \ifdim\csname#1\endcsname<\csname#2\endcsname\relax
543       \forest@sort@cmp@lt
544     \else
545       \forest@sort@cmp@eq
546     \fi
547   \fi
548 }

```

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

```

549 \def\forest@sort@cmptwodimcs#1#2#3#4{%
550   \ifdim\csname#1\endcsname>\csname#3\endcsname\relax
551     \forest@sort@cmp@gt
552   \else
553     \ifdim\csname#1\endcsname<\csname#3\endcsname\relax
554       \forest@sort@cmp@lt
555     \else
556       \ifdim\csname#2\endcsname>\csname#4\endcsname\relax

```

```

557     \forest@sort@cmp@gt
558   \else
559     \ifdim\csname#2\endcsname<\csname#4\endcsname\relax
560     \forest@sort@cmp@lt
561   \else
562     \forest@sort@cmp@eq
563   \fi
564 \fi
565 \fi
566 \fi
567 }

```

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).

```

568 \def\forest@reversearray#1#2#3{%
569   \let\forest@sort@let#1%
570   \c@pgf@countc=#2
571   \c@pgf@countd=#3
572   \advance\c@pgf@countd -1
573   \safeloop
574   \ifnum\c@pgf@countc<\c@pgf@countd\relax
575     \forest@sort@xch{\the\c@pgf@countc}{\the\c@pgf@countd}%
576     \advance\c@pgf@countc 1
577     \advance\c@pgf@countd -1
578   \saferepeat
579 }

```

5 The bracket representation parser

5.1 The user interface macros

Settings.

```

580 \def\bracketset#1{\pgfkeys{/bracket}{#1}}%
581 \bracketset{%
582   /bracket/.is family,
583   /handlers/.let/.style={\pgfkeyscurrentpath/.code={\let#1#1}},
584   opening bracket/.let=\bracket@openingBracket,
585   closing bracket/.let=\bracket@closingBracket,
586   action character/.let=\bracket@actionCharacter,
587   opening bracket=[,
588   closing bracket=],
589   action character,
590   new node/.code n args={3}{% #1=preamble, #2=node spec, #3=cs receiving the id
591     \forest@node@new#3%
592     \forest@set{#3}{given options}{\forest@contentto=\unexpanded{#2}}%
593     \ifblank{#1}{}%
594     \forest@set{preamble}{#1}%
595   }%
596 },
597   set afterthought/.code 2 args={% #1=node id, #2=afterthought
598     \ifblank{#2}{\forest@appto{#1}{given options}{,afterthought={#2}}}%
599   }
600 }

```

`\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.)

```

601 \newtoks\bracket@content
602 \newtoks\bracket@afterthought

```

```

603 \def\bracketParse#1#2={%
604   \def\bracketEndParsingHook{#1}%
605   \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.)

```

606 \bracket@content={}%
607 \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.

```

608 \let\bracket@state\bracket@state@starting
609 \bracket@ignorespacestrue

```

By default, don’t expand anything.

```

610 \bracket@expandtokensfalse

```

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

```

611 \def\bracket@parentNode{0}%
612 \def\bracket@rootNode{0}%
613 \def\bracket@newNode{0}%
614 \def\bracket@afterthoughtNode{0}%

```

Finally, we start the parser.

```

615 \bracket@Parse
616 }

```

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.

```

617 \def\bracketResume{\bracket@Parse}%

```

5.2 Parsing

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

```

618 \def\bracket@Parse{%
619   \futurelet\bracket@next@token\bracket@Parse@checkForSpace
620 }
621 \def\bracket@Parse@checkForSpace{%
622   \expandafter\ifx\space\bracket@next@token\@escapeif{%
623     \ifbracket@ignorespaces\else
624       \bracket@haveSpace>true
625     \fi
626     \expandafter\bracket@Parse\romannumeral-‘0%
627   }\else\@escapeif{%
628     \bracket@Parse@maybeexpand
629   }\fi
630 }

```

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`.

```

631 \newif\ifbracket@expandtokens
632 \def\bracket@Parse@maybeexpand{%
633   \ifbracket@expandtokens\@escapeif{%
634     \expandafter\bracket@Parse@peekAhead\romannumeral-‘0%
635   }\else\@escapeif{%
636     \bracket@Parse@peekAhead
637   }\fi
638 }

```

We then look ahead to see what's coming.

```
639 \def\bracket@Parse@peekAhead{%
640   \futurelet\bracket@next@token\bracket@Parse@checkForTeXGroup
641 }
```

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

```
642 \def\bracket@Parse@checkForTeXGroup{%
643   \ifx\bracket@next@token\bgroup%
644     \@escapeif{\bracket@Parse@appendGroup}%
645   \else
646     \@escapeif{\bracket@Parse@token}%
647   \fi
648 }
```

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.

```
649 \long\def\bracket@Parse@token#1{%
650   \ifx#1\bracket@openingBracket
651     \@escapeif{\bracket@Parse@openingBracketFound}%
652   \else
653     \@escapeif{%
654       \ifx#1\bracket@closingBracket
655         \@escapeif{\bracket@Parse@closingBracketFound}%
656       \else
657         \@escapeif{%
658           \ifx#1\bracket@actionCharacter
659             \@escapeif{\futurelet\bracket@next@token\bracket@Parse@actionCharacterFound}%
660           \else
661             \@escapeif{\bracket@Parse@appendToken#1}%
662           \fi
663         }%
664       \fi
665     }%
666   \fi
667 }
```

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

```
668 \newif\ifbracket@haveSpace
669 \newif\ifbracket@ignorespaces
670 \def\bracket@Parse@appendSpace{%
671   \ifbracket@haveSpace
672     \ifcase\bracket@state\relax
673       \eapptotoks\bracket@content\space
674     \or
675       \eapptotoks\bracket@afterthought\space
676     \or
677       \eapptotoks\bracket@afterthought\space
678     \fi
679   \bracket@haveSpacefalse
680 \fi
681 }
682 \long\def\bracket@Parse@appendToken#1{%
683   \bracket@Parse@appendSpace
684   \ifcase\bracket@state\relax
685     \lapptotoks\bracket@content{#1}%
686   \or
687     \lapptotoks\bracket@afterthought{#1}%
688   \or
689     \lapptotoks\bracket@afterthought{#1}%
690   \fi
691 }
```

```

690 \fi
691 \bracket@ignorespacesfalse
692 \bracket@Parse
693 }
694 \def\bracket@Parse@appendGroup#1{%
695 \bracket@Parse@appendSpace
696 \ifcase\bracket@state\relax
697 \apptotoks\bracket@content{{#1}}%
698 \or
699 \apptotoks\bracket@afterthought{{#1}}%
700 \or
701 \apptotoks\bracket@afterthought{{#1}}%
702 \fi
703 \bracket@ignorespacesfalse
704 \bracket@Parse
705 }

```

Declare states.

```

706 \def\bracket@state@inContent{0}
707 \def\bracket@state@inAfterthought{1}
708 \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.)

```

709 \def\bracket@Parse@openingBracketFound{%
710 \bracket@haveSpacefalse
711 \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.

```

712 \escapeif{%
713 \bracket@createNode
714 \ifnum\bracket@parentNode=0 \else
715 \forest@node@Append{\bracket@parentNode}{\bracket@newNode}%
716 \fi
717 \let\bracket@parentNode\bracket@newNode
718 \bracket@Parse
719 }%
720 \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.

```

721 \escapeif{%
722 \bracket@addAfterthought
723 \let\bracket@state\bracket@state@inContent
724 \bracket@Parse
725 }%
726 \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.

```

727 \escapeif{%
728 \let\bracket@state\bracket@state@inContent
729 \bracket@Parse
730 }%
731 \fi
732 }

```



```

733 \def\bracket@Parse@closingBracketFound{%
734   \bracket@haveSpacefalse
735   \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.

```

736   \@escapeif{%
737     \bracket@createNode
738     \ifnum\bracket@parentNode=0
739       \@escapeif\bracket@endParsingHook
740     \else
741       \@escapeif{%
742         \let\bracket@afterthoughtNode\bracket@newNode
743         \let\bracket@state\bracket@state@inAfterthought
744         \forest@node@Append{\bracket@parentNode}{\bracket@newNode}%
745         \bracket@Parse
746       }%
747     \fi
748   }%
749   \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.

```

750   \@escapeif{%
751     \bracket@addAfterthought
752     \let\bracket@afterthoughtNode\bracket@parentNode
753     \edef\bracket@parentNode{\forestOve{\bracket@parentNode}{@parent}}%
754     \ifnum\bracket@parentNode=0
755       \expandafter\bracket@endParsingHook
756     \else
757       \expandafter\bracket@Parse
758     \fi
759   }%
760   \else % starting

```

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

```

761   \PackageError{forest}{You're attempting to start a bracket representation
762     with a closing bracket}{}%
763   \fi
764 }

```

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

```

765 \def\bracket@Parse@actionCharacterFound{%
  If a braced expression follows, its contents will be fully expanded.
766   \ifx\bracket@next@token\bgroup\@escapeif{%
767     \bracket@Parse@action@expandgroup
768   }\else\@escapeif{%
769     \bracket@Parse@action@notagroup
770   }\fi
771 }
772 \def\bracket@Parse@action@expandgroup#1{%
773   \edef\bracket@Parse@action@expandgroup@macro{#1}%
774   \expandafter\bracket@Parse\bracket@Parse@action@expandgroup@macro
775 }
776 \let\bracket@action@fullyexpandCharacter+
777 \let\bracket@action@dontexpandCharacter-
778 \let\bracket@action@executeCharacter!
779 \def\bracket@Parse@action@notagroup#1{%

```

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

```
780 \ifx#1\bracket@action@fullyexpandCharacter\@escapeif{%
781   \bracket@expandtokenstrue\bracket@Parse
782 }\else\@escapeif{%
```

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

```
783 \ifx#1\bracket@action@dontexpandCharacter\@escapeif{%
784   \bracket@expandtokensfalse\bracket@Parse
785 }\else\@escapeif{%
```

Inhibit expansion of the next token.

```
786 \ifx#10\@escapeif{%
787   \bracket@Parse@appendToken
788 }\else\@escapeif{%
```

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

```
789 \ifx#1\bracket@actionCharacter
790 \else\@escapeif{%
```

Anything else will be expanded once.

```
791 \expandafter\bracket@Parse#1%
792 }\fi
793 }\fi
794 }\fi
795 }\fi
796 }
```

5.3 The tree-structure interface

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

```
797 \def\bracket@createNode{%
798   \ifnum\bracket@rootNode=0
799     % root node
800     \bracketset{new node/.expanded=%
801       {\the\bracket@afterthought}%
802       {\the\bracket@content}%
803       \noexpand\bracket@newNode
804     }%
805     \bracket@afterthought={}%
806     \let\bracket@rootNode\bracket@newNode
807     \expandafter\let\bracket@saveRootNodeTo\bracket@newNode
808   \else
809     % other nodes
810     \bracketset{new node/.expanded=%
811       {\the\bracket@afterthought}%
812       {\the\bracket@content}%
813       \noexpand\bracket@newNode
814     }%
815   \fi
816   \bracket@content={}%
817 }
```

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

```
818 \def\bracket@addAfterthought{%
819   \bracketset{%
820     set afterthought/.expanded={\bracket@afterthoughtNode}{\the\bracket@afterthought}%
821   }%
822   \bracket@afterthought={}%

```

823 }

6 Nodes

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

6.1 Option setting and retrieval

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

```

824 % full expansion expands precisely to the value
825 \def\forestov#1{\expandafter\expandafter\expandafter\expandonce
826   \pgfkeysvalueof{/forest/@node/\forest@cn/#1}}
827 % full expansion expands all the way
828 \def\forestove#1{\pgfkeysvalueof{/forest/@node/\forest@cn/#1}}
829 % full expansion expands to the cs holding the value
830 \def\forestom#1{\expandafter\expandonce\expandafter{\pgfkeysvalueof{/forest/@node/\forest@cn/#1}}}
831 \def\forestoget#1#2{\pgfkeysgetvalue{/forest/@node/\forest@cn/#1}{#2}}
832 \def\forestolet#1#2{\pgfkeyslet{/forest/@node/\forest@cn/#1}{#2}}
833 \def\forestocslet#1#2{%
834   \edef\forest@marshal{%
835     \noexpand\pgfkeyslet{/forest/@node/\forest@cn/#1}{\expandonce{\csname#2\endcsname}}}%
836   }\forest@marshal
837 }
838 \def\forestoset#1#2{\pgfkeyssetvalue{/forest/@node/\forest@cn/#1}{#2}}
839 \def\forestoeset#1#2{%
840   \edef\forest@option@temp{%
841     \noexpand\pgfkeyssetvalue{/forest/@node/\forest@cn/#1}{#2}%
842   }\forest@option@temp
843 }
844 \def\forestooppto#1#2{%
845   \forestoeset{#1}{\forestov{#1}\unexpanded{#2}}}%
846 }
847 \def\forestoiifdefined#1#2#3{%
848   \pgfkeysifdefined{/forest/@node/\forest@cn/#1}{#2}{#3}%
849 }

```

User macros for retrieving node options of the current node.

```

850 \let\forestoption\forestov
851 \let\foresteooption\forestove

```

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

```

852 \def\forestov#1#2{\expandafter\expandafter\expandafter\expandonce
853   \pgfkeysvalueof{/forest/@node/#1/#2}}
854 \def\forestove#1#2{\pgfkeysvalueof{/forest/@node/#1/#2}}
855 % full expansion expands to the cs holding the value
856 \def\forestom#1#2{\expandafter\expandonce\expandafter{\pgfkeysvalueof{/forest/@node/#1/#2}}}
857 \def\forestoget#1#2#3{\pgfkeysgetvalue{/forest/@node/#1/#2}{#3}}
858 \def\forestolet#1#2#3{\pgfkeyslet{/forest/@node/#1/#2}{#3}}
859 \def\forestolet#1#2#3{\pgfkeyslet{/forest/@node/#1/#2}{#3}}
860 \def\forestocslet#1#2#3{%
861   \edef\forest@marshal{%
862     \noexpand\pgfkeyslet{/forest/@node/#1/#2}{\expandonce{\csname#3\endcsname}}}%
863   }\forest@marshal
864 }
865 \def\forestoset#1#2#3{\pgfkeyssetvalue{/forest/@node/#1/#2}{#3}}
866 \def\forestoeset#1#2#3{%
867   \edef\forestoption@temp{%
868     \noexpand\pgfkeyssetvalue{/forest/@node/#1/#2}{#3}%
869   }\forestoption@temp

```

```

870 }
871 \def\forest0appto#1#2#3{%
872   \forest0eset{#1}{#2}{\forest0v{#1}{#2}\unexpanded{#3}}%
873 }
874 \def\forest0eappto#1#2#3{%
875   \forest0eset{#1}{#2}{\forest0v{#1}{#2}#3}%
876 }
877 \def\forest0preto#1#2#3{%
878   \forest0eset{#1}{#2}{\unexpanded{#3}\forest0v{#1}{#2}}%
879 }
880 \def\forest0epreto#1#2#3{%
881   \forest0eset{#1}{#2}{#3\forest0v{#1}{#2}}%
882 }
883 \def\forest0ifdefined#1#2#3#4{%
884   \pgfkeysifdefined{/forest/@node/#1/#2}{#3}{#4}%
885 }
886 \def\forest0let0#1#2#3#4{% option #2 of node #1 <-- option #4 of node #3
887   \forest0get{#3}{#4}\forestoption@temp
888   \forest0let{#1}{#2}\forestoption@temp}
889 \def\forest0leto#1#2#3{%
890   \forest0get{#3}\forestoption@temp
891   \forest0let{#1}{#2}\forestoption@temp}
892 \def\forest0tolet0#1#2#3{%
893   \forest0get{#2}{#3}\forestoption@temp
894   \forest0let{#1}\forestoption@temp}
895 \def\forest0toleto#1#2{%
896   \forest0get{#2}\forestoption@temp
897   \forest0let{#1}\forestoption@temp}

  Macros for retrieving/setting registers.

898 % full expansion expands precisely to the value
899 \def\forestrv#1{\expandafter\expandafter\expandafter\expandonce
900   \pgfkeysvalueof{/forest/@node/register/#1}}
901 % full expansion expands all the way
902 \def\forestrv#1{\pgfkeysvalueof{/forest/@node/register/#1}}
903 % full expansion expands to the cs holding the value
904 \def\forestrm#1{\expandafter\expandonce\expandafter{\pgfkeysvalueof{/forest/@node/register/#1}}}
905 \def\forestrget#1#2{\pgfkeysgetvalue{/forest/@node/register/#1}{#2}}
906 \def\forestrlet#1#2{\pgfkeyslet{/forest/@node/register/#1}{#2}}
907 \def\forestrcslet#1#2{%
908   \edef\forest@marshal{%
909     \noexpand\pgfkeyslet{/forest/@node/register/#1}{\expandonce{\csname#2\endcsname}}%
910   }\forest@marshal
911 }
912 \def\forestrset#1#2{\pgfkeyssetvalue{/forest/@node/register/#1}{#2}}
913 \def\forestreset#1#2{%
914   \edef\forest@option@temp{%
915     \noexpand\pgfkeyssetvalue{/forest/@node/register/#1}{#2}%
916   }\forest@option@temp
917 }
918 \def\forestrappto#1#2{%
919   \forestreset{#1}{\forestrv{#1}\unexpanded{#2}}%
920 }
921 \def\forestrpreto#1#2{%
922   \forestreset{#1}{\unexpanded{#2}\forestrv{#1}}%
923 }
924 \def\forestrifdefined#1#2#3{%
925   \pgfkeysifdefined{/forest/@node/register/#1}{#2}{#3}%
926 }

  User macros for retrieving node options of the current node.
927 \def\forestregister#1{\forestrv{#1}}

```

```

928 \def\forestregister#1{\forestrve{#1}}
Node initialization. Node option declarations append to \forest@node@init.
929 \def\forest@node@init{%
930   \forestoset{@parent}{0}%
931   \forestoset{@previous}{0}% previous sibling
932   \forestoset{@next}{0}% next sibling
933   \forestoset{@first}{0}% primary child
934   \forestoset{@last}{0}% last child
935 }
936 \def\forest@node@init#1{%
937   \pgfkeysgetvalue{/forest/#1}\forest@node@temp
938   \forest@node@init#1\forest@node@temp
939 }
940 \newcount\forest@node@maxid
941 \def\forest@node@new#1{% #1 = cs receiving the new node id
942   \advance\forest@node@maxid1
943   \forest@node@init#1\forest@node@maxid}%
944   \forest@node@init
945   \forest@node@setname{node@\forest@cn}%
946   \forest@node@initializefromstandardnode
947   \edef#1{\forest@cn}%
948 }%
949 }
950 \let\forest@node@orig\forest@node@init
951 \def\forest@node@copy#1#2{% #1=from node id, #2=cs receiving the new node id
952   \advance\forest@node@maxid1
953   \def\forest@node@init##1{\ifstrequal{##1}{name}{\forestoset{name}{node@\forest@cn}}{\forest@node@init}}
954   \forest@node@init#1\forest@node@maxid}%
955   \forest@node@init
956   \forest@node@setname{\forest@node@copy@name@template{\forest@node@copy@name@template{#1}{name}}}%
957   \edef#2{\forest@cn}%
958 }%
959 \let\forest@node@orig\forest@node@init
960 }
961 \forestset{
962   copy name template/.code={\def\forest@node@copy@name@template##1{#1}},
963   copy name template/.default={node@\forest@node@maxid},
964   copy name template
965 }
966 \def\forest@tree@copy#1#2{% #1=from node id, #2=cs receiving the new node id
967   \forest@node@copy{#1}\forest@node@copy@temp@id
968   \forest@node@init#1\forest@node@copy@temp@id}%
969   \expandafter\forest@tree@copy\expandafter{\forest@node@copy@temp@id}{#1}%
970   \edef#2{\forest@cn}%
971 }%
972 }
973 \def\forest@tree@copy@node@#1#2{%
974   \forest@node@foreachchild{#2}{%
975     \expandafter\forest@tree@copy\expandafter{\forest@cn}\forest@node@copy@temp@childid
976     \forest@node@append{#1}{\forest@node@copy@temp@childid}%
977   }%
978 }
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.
979 \def\forest@cn{0}
980 \forest@node@init

```

6.2 Tree structure

Node insertion/removal.

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

```

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

```

```

1035 \forestOget{#1}{@last}\forest@node@temp
1036 \forestOeset{#1}{@last}{#2}%
1037 \forestOeset{#2}{@previous}\forest@node@temp}%
1038 \ifnum\forest@node@temp=0
1039 \forestOeset{#1}{@first}{#2}%
1040 \else
1041 \forestOeset{\forest@node@temp}{@next}{#2}%
1042 \fi
1043 }
1044 \def\forest@node@Remove@#1{%
1045 \forestOget{#1}{@parent}\forest@node@temp@parent
1046 \ifnum\forest@node@temp@parent=0
1047 \else
1048 \forestOget{#1}{@previous}\forest@node@temp@previous
1049 \forestOget{#1}{@next}\forest@node@temp@next
1050 \ifnum\forest@node@temp@previous=0
1051 \forestOeset{\forest@node@temp@parent}{@first}\forest@node@temp@next}%
1052 \else
1053 \forestOeset{\forest@node@temp@previous}{@next}\forest@node@temp@next}%
1054 \fi
1055 \ifnum\forest@node@temp@next=0
1056 \forestOeset{\forest@node@temp@parent}{@last}\forest@node@temp@previous}%
1057 \else
1058 \forestOeset{\forest@node@temp@next}{@previous}\forest@node@temp@previous}%
1059 \fi
1060 \forestOset{#1}{@parent}{0}%
1061 \forestOset{#1}{@previous}{0}%
1062 \forestOset{#1}{@next}{0}%
1063 \fi
1064 }

```

Do some stuff and return to the current node.

```

1065 \def\forest@forthis#1{%
1066 \edef\forest@node@marshal{\unexpanded{#1}\def\noexpand\forest@cn}%
1067 \expandafter\forest@node@marshal\expandafter{\forest@cn}%
1068 }
1069 \def\forest@fornode#1#2{%
1070 \edef\forest@node@marshal{\edef\noexpand\forest@cn{#1}\unexpanded{#2}\def\noexpand\forest@cn}%
1071 \expandafter\forest@node@marshal\expandafter{\forest@cn}%
1072 }

```

Looping methods: children.

```

1073 \def\forest@node@foreachchild#1{\forest@node@Foreachchild{\forest@cn}{#1}}
1074 \def\forest@node@Foreachchild#1#2{%
1075 \forest@fornode{\forestOve{#1}{@first}}{\forest@node@@forselfandfollowingsiblings{#2}}%
1076 }
1077 \def\forest@node@@forselfandfollowingsiblings#1{%
1078 \ifnum\forest@cn=0
1079 \else
1080 \forest@forthis{#1}%
1081 \@escapeif{%
1082 \edef\forest@cn{\forestove{@next}}%
1083 \forest@node@@forselfandfollowingsiblings{#1}%
1084 }%
1085 \fi
1086 }
1087 \def\forest@node@@forselfandfollowingsiblings@reversed#1{%
1088 \ifnum\forest@cn=0
1089 \else
1090 \@escapeif{%
1091 \edef\forest@marshal{%
1092 \noexpand\def\noexpand\forest@cn{\forestove{@next}}%

```

```

1093     \noexpand\forest@node@@forselfandfollowingsiblings@reversed{\unexpanded{#1}}%
1094     \noexpand\forest@fornode{\forest@cn}{\unexpanded{#1}}%
1095   }\forest@marshal
1096 }%
1097 \fi
1098 }
1099 \def\forest@node@foreachchild@reversed#1{\forest@node@Foreachchild@reversed{\forest@cn}{#1}}
1100 \def\forest@node@Foreachchild@reversed#1#2{%
1101   \forest@fornode{\forest@ve{#1}{@last}}{\forest@node@@forselfandprecedingsiblings@reversed{#2}}%
1102 }
1103 \def\forest@node@@forselfandprecedingsiblings@reversed#1{%
1104   \ifnum\forest@cn=0
1105   \else
1106     \forest@forthis{#1}%
1107     \@escapeif{%
1108       \edef\forest@cn{\forest@ve{@previous}}%
1109       \forest@node@@forselfandprecedingsiblings@reversed{#1}%
1110     }%
1111   \fi
1112 }
1113 \def\forest@node@@forselfandprecedingsiblings#1{%
1114   \ifnum\forest@cn=0
1115   \else
1116     \@escapeif{%
1117       \edef\forest@marshal{%
1118         \noexpand\def\noexpand\forest@cn{\forest@ve{@previous}}%
1119         \noexpand\forest@node@@forselfandprecedingsiblings{\unexpanded{#1}}%
1120         \noexpand\forest@fornode{\forest@cn}{\unexpanded{#1}}%
1121       }\forest@marshal
1122     }%
1123   \fi
1124 }

```

Looping methods: (sub)tree and descendants.

```

1125 \def\forest@node@@foreach#1#2#3#4{%
1126   % #1 = do what
1127   % #2 = do that -1=before,1=after processing children
1128   % #3 & #4: normal or reversed order of children?
1129   % #3 = @first/@last
1130   % #4 = \forest@node@@forselfandfollowingsiblings / \forest@node@@forselfandprecedingsiblings@reversed
1131   \ifnum#2<0 \forest@forthis{#1}\fi
1132   \ifnum\forest@ve{#3}=0
1133   \else\@escapeif{%
1134     \forest@forthis{%
1135       \edef\forest@cn{\forest@ve{#3}}%
1136       #4{\forest@node@@foreach{#1}{#2}{#3}{#4}}%
1137     }%
1138   }\fi
1139   \ifnum#2>0 \forest@forthis{#1}\fi
1140 }
1141 \def\forest@node@foreach#1{%
1142   \forest@node@@foreach{#1}{-1}{@first}{\forest@node@@forselfandfollowingsiblings}
1143 \def\forest@node@Foreach#1#2{%
1144   \forest@fornode{#1}{\forest@node@@foreach{#2}{-1}{@first}{\forest@node@@forselfandfollowingsiblings}}
1145 \def\forest@node@foreach@reversed#1{%
1146   \forest@node@@foreach{#1}{-1}{@last}{\forest@node@@forselfandprecedingsiblings@reversed}
1147 \def\forest@node@Foreach@reversed#1#2{%
1148   \forest@fornode{#1}{\forest@node@@foreach{#2}{-1}{@last}{\forest@node@@forselfandprecedingsiblings@reversed}
1149 \def\forest@node@foreach@childrenfirst#1{%
1150   \forest@node@@foreach{#1}{1}{@first}{\forest@node@@forselfandfollowingsiblings}
1151 \def\forest@node@Foreach@childrenfirst#1#2{%

```



```

1152 \forest@fornode{#1}{\forest@node@@foreach{#2}{1}{@first}{\forest@node@@forselfandfollowingsiblings}}
1153 \def\forest@node@foreach@childrenfirst@reversed#1{%
1154 \forest@node@@foreach{#1}{1}{@last}{\forest@node@@forselfandprecedingsiblings@reversed}}
1155 \def\forest@node@Foreach@childrenfirst@reversed#1#2{%
1156 \forest@fornode{#1}{\forest@node@@foreach{#2}{1}{@last}{\forest@node@@forselfandprecedingsiblings@reversed}}
1157 \def\forest@node@foreachdescendant#1{%
1158 \forest@node@foreachchild{\forest@node@@foreach{#1}{-1}{@first}{\forest@node@@forselfandfollowingsiblings}}
1159 \def\forest@node@Foreachdescendant#1#2{%
1160 \forest@node@Foreachchild{#1}{\forest@node@@foreach{#2}{-1}{@first}{\forest@node@@forselfandfollowingsiblings}}
1161 \def\forest@node@foreachdescendant@reversed#1{%
1162 \forest@node@foreachchild@reversed{\forest@node@@foreach{#1}{-1}{@last}{\forest@node@@forselfandprecedingsiblings@reversed}}
1163 \def\forest@node@Foreachdescendant@reversed#1#2{%
1164 \forest@node@Foreachchild@reversed{#1}{\forest@node@@foreach{#2}{-1}{@last}{\forest@node@@forselfandprecedingsiblings@reversed}}
1165 \def\forest@node@foreachdescendant@childrenfirst#1{%
1166 \forest@node@foreachchild{\forest@node@@foreach{#1}{1}{@first}{\forest@node@@forselfandfollowingsiblings}}
1167 \def\forest@node@Foreachdescendant@childrenfirst#1#2{%
1168 \forest@node@Foreachchild{#1}{\forest@node@@foreach{#2}{1}{@first}{\forest@node@@forselfandfollowingsiblings}}
1169 \def\forest@node@foreachdescendant@childrenfirst@reversed#1{%
1170 \forest@node@foreachchild@reversed{\forest@node@@foreach{#1}{1}{@last}{\forest@node@@forselfandprecedingsiblings@reversed}}
1171 \def\forest@node@Foreachdescendant@childrenfirst@reversed#1#2{%
1172 \forest@node@Foreachchild@reversed{#1}{\forest@node@@foreach{#2}{1}{@last}{\forest@node@@forselfandprecedingsiblings@reversed}}

```

Looping methods: breadth-first.

```

1173 \def\forest@node@foreach@breadthfirst#1#2{% #1 = max level, #2 = code
1174 \forest@node@Foreach@breadthfirst@{\forest@cn}{@first}{@next}{#1}{#2}}
1175 \def\forest@node@foreach@breadthfirst@reversed#1#2{% #1 = max level, #2 = code
1176 \forest@node@Foreach@breadthfirst@{\forest@cn}{@last}{@previous}{#1}{#2}}
1177 \def\forest@node@Foreach@breadthfirst#1#2#3{% #1 = node id, #2 = max level, #3 = code
1178 \forest@node@Foreach@breadthfirst@{#1}{@first}{@next}{#2}{#3}}
1179 \def\forest@node@Foreach@breadthfirst@reversed#1#2#3{% #1 = node id, #2 = max level, #3 = code
1180 \forest@node@Foreach@breadthfirst@{#1}{@last}{@previous}{#2}{#3}}
1181 \def\forest@node@Foreach@breadthfirst@#1#2#3#4#5{%
1182 % #1 = root node,
1183 % #2 = @first/@last, #3 = @next/@previous (must be in sync with #2),
1184 % #4 = max level (< 0 means infinite)
1185 % #5 = code to execute at each node
1186 \forest@node@Foreach@breadthfirst@processqueue{#1},{#2}{#3}{#4}{#5}%
1187 }
1188 \def\forest@node@Foreach@breadthfirst@processqueue#1#2#3#4#5{%
1189 % #1 = queue,
1190 % #2 = @first/@last, #3 = @next/@previous (must be in sync with #2),
1191 % #4 = max level (< 0 means infinite)
1192 % #5 = code to execute at each node
1193 \ifstrempy{#1}{-}{%
1194 \forest@node@Foreach@breadthfirst@processqueue@#1\forest@node@Foreach@breadthfirst@processqueue@
1195 {#2}{#3}{#4}{#5}%
1196 }%
1197 }
1198 \def\forest@node@Foreach@breadthfirst@processqueue@#1,#2\forest@node@Foreach@breadthfirst@processqueue@#3#4#5{%
1199 % #1 = first,
1200 % #2 = rest,
1201 % #3 = @first/@last, #4 = next/previous (must be in sync with #2),
1202 % #5 = max level (< 0 means infinite)
1203 % #6 = code to execute at each node
1204 \forest@fornode{#1}{%
1205 #6%
1206 \ifnum#5<0
1207 \forest@node@getlistofchildren\forest@temp{#3}{#4}%
1208 \else
1209 \ifnum\forest@tove{level}>#5\relax
1210 \def\forest@temp{%

```

```

1211     \else
1212     \forest@node@getlistofchildren\forest@temp{#3}{#4}%
1213     \fi
1214     \fi
1215     \edef\forest@marshal{%
1216     \noexpand\forest@node@Foreach@breadthfirst@processqueue{\unexpanded{#2}\forest@temp}%
1217     {#3}{#4}{#5}{\unexpanded{#6}}}%
1218     }\forest@marshal
1219 }%
1220 }
1221 \def\forest@node@getlistofchildren#1#2#3{% #1 = list cs, #2 = @first/@last, #3 = @next/@previous
1222 \forest@node@Getlistofchildren{\forest@cn}{#1}{#2}{#3}%
1223 }
1224 \def\forest@node@Getlistofchildren#1#2#3#4{% #1 = node, #2 = list cs, #3 = @first/@last, #4 = @next/@previous
1225 \def#2{}%
1226 \ifnum\forestove{#3}=0
1227 \else
1228 \eappto#2{\forestOve{#1}{#3}},}%
1229 \@escapeif{%
1230 \edef\forest@marshal{%
1231 \noexpand\forest@node@Getlistofchildren@{\forestOve{#1}{#3}}\noexpand#2{#4}%
1232 }\forest@marshal
1233 }%
1234 \fi
1235 }
1236 \def\forest@node@Getlistofchildren@#1#2#3{% #1 = node, #2 = list cs, #3 = @next/@previous
1237 \ifnum\forestOve{#1}{#3}=0
1238 \else
1239 \eappto#2{\forestOve{#1}{#3}},}%
1240 \@escapeif{%
1241 \edef\forest@marshal{%
1242 \noexpand\forest@node@Getlistofchildren@{\forestOve{#1}{#3}}\noexpand#2{#3}%
1243 }\forest@marshal
1244 }%
1245 \fi
1246 }

Compute n, n', n children and level.
1247 \def\forest@node@Compute@numeric@ts@info@#1{%
1248 \forest@node@Foreach{#1}{\forest@node@@compute@numeric@ts@info}%
1249 \ifnum\forestOve{#1}{@parent}=0
1250 \else
1251 \forest@for@node{#1}{\forest@node@@compute@numeric@ts@info@nbar}%
1252 % hack: the parent of the node we called the update for gets +1 for n_children
1253 \edef\forest@node@temp{\forestOve{#1}{@parent}}%
1254 \forestOset{\forest@node@temp}{n children}{%
1255 \number\numexpr\forestOve{\forest@node@temp}{n children}-1%
1256 }%
1257 \fi
1258 \forest@node@Foreachdescendant{#1}{\forest@node@@compute@numeric@ts@info@nbar}%
1259 }
1260 \def\forest@node@@compute@numeric@ts@info{%
1261 \forestoset{n children}{0}%
1262 %
1263 \edef\forest@node@temp{\forestove{@previous}}%
1264 \ifnum\forest@node@temp=0
1265 \forestoset{n}{1}%
1266 \else
1267 \forestoeset{n}{\number\numexpr\forestOve{\forest@node@temp}{n}+1}%
1268 \fi
1269 %

```

```

1270 \edef\forest@node@temp{\forestove{@parent}}%
1271 \ifnum\forest@node@temp=0
1272   \forestoset{n}{0}%
1273   \forestoset{n'}{0}%
1274   \forestoset{level}{0}%
1275 \else
1276   \forestOset{\forest@node@temp}{n children}{%
1277     \number\numexpr\forestOve{\forest@node@temp}{n children}+1%
1278   }%
1279   \forestoeset{level}{%
1280     \number\numexpr\forestOve{\forest@node@temp}{level}+1%
1281   }%
1282 \fi
1283 }
1284 \def\forest@node@@compute@numeric@ts@info@nbar{%
1285   \forestoeset{n'}{\number\numexpr\forestOve{\forestove{@parent}}{n children}-\forestove{n}+1}%
1286 }
1287 \def\forest@node@compute@numeric@ts@info#1{%
1288   \expandnumberarg\forest@node@Compute@numeric@ts@info@{\forest@cn}%
1289 }
1290 \def\forest@node@Compute@numeric@ts@info#1{%
1291   \expandnumberarg\forest@node@Compute@numeric@ts@info@{#1}%
1292 }

```

Tree structure queries.

```

1293 \def\forest@node@rootid{%
1294   \expandnumberarg\forest@node@Rootid{\forest@cn}%
1295 }
1296 \def\forest@node@Rootid#1{% #1=node
1297   \ifnum\forestOve{#1}{@parent}=0
1298     #1%
1299   \else
1300     \@escapeif{\expandnumberarg\forest@node@Rootid{\forestOve{#1}{@parent}}}%
1301   \fi
1302 }
1303 \def\forest@node@nthchildid#1{% #1=n
1304   \ifnum#1<1
1305     0%
1306   \else
1307     \expandnumberarg\forest@node@nthchildid@{\number\forestove{@first}}{#1}%
1308   \fi
1309 }
1310 \def\forest@node@nthchildid@#1#2{%
1311   \ifnum#1=0
1312     0%
1313   \else
1314     \ifnum#2>1
1315       \@escapeifif{\expandtwonumberargs
1316         \forest@node@nthchildid@{\forestOve{#1}{@next}}{\numexpr#2-1}}%
1317     \else
1318       #1%
1319     \fi
1320   \fi
1321 }
1322 \def\forest@node@nbarthchildid#1{% #1=n
1323   \expandnumberarg\forest@node@nbarthchildid@{\number\forestove{@last}}{#1}%
1324 }
1325 \def\forest@node@nbarthchildid@#1#2{%
1326   \ifnum#1=0
1327     0%
1328   \else

```

```

1329 \ifnum#2>1
1330 \escapeifif{\expandtwonumberargs
1331 \forest@node@nbarthchildid@{\forestOve{#1}{@previous}}{\numexpr#2-1}}%
1332 \else
1333 #1%
1334 \fi
1335 \fi
1336 }
1337 \def\forest@node@nornbarthchildid#1{%
1338 \ifnum#1>0
1339 \forest@node@nthchildid{#1}%
1340 \else
1341 \ifnum#1<0
1342 \forest@node@nbarthchildid{-#1}%
1343 \else
1344 \forest@node@nornbarthchildid@error
1345 \fi
1346 \fi
1347 }
1348 \def\forest@node@nornbarthchildid@error{%
1349 \PackageError{forest}{In \string\forest@node@nornbarthchildid, n should !=0}{}%
1350 }
1351 \def\forest@node@previousleafid{%
1352 \expandnumberarg\forest@node@Previousleafid{\forest@cn}%
1353 }
1354 \def\forest@node@Previousleafid#1{%
1355 \ifnum\forestOve{#1}{@previous}=0
1356 \escapeif{\expandnumberarg\forest@node@previousleafid@Goup{#1}}%
1357 \else
1358 \expandnumberarg\forest@node@previousleafid@Godown{\forestOve{#1}{@previous}}%
1359 \fi
1360 }
1361 \def\forest@node@previousleafid@Goup#1{%
1362 \ifnum\forestOve{#1}{@parent}=0
1363 \PackageError{forest}{get previous leaf: this is the first leaf}{}%
1364 \else
1365 \escapeif{\expandnumberarg\forest@node@Previousleafid{\forestOve{#1}{@parent}}}%
1366 \fi
1367 }
1368 \def\forest@node@previousleafid@Godown#1{%
1369 \ifnum\forestOve{#1}{@last}=0
1370 #1%
1371 \else
1372 \escapeif{\expandnumberarg\forest@node@previousleafid@Godown{\forestOve{#1}{@last}}}%
1373 \fi
1374 }
1375 \def\forest@node@nextleafid{%
1376 \expandnumberarg\forest@node@Nextleafid{\forest@cn}%
1377 }
1378 \def\forest@node@Nextleafid#1{%
1379 \ifnum\forestOve{#1}{@next}=0
1380 \escapeif{\expandnumberarg\forest@node@nextleafid@Goup{#1}}%
1381 \else
1382 \expandnumberarg\forest@node@nextleafid@Godown{\forestOve{#1}{@next}}%
1383 \fi
1384 }
1385 \def\forest@node@nextleafid@Goup#1{%
1386 \ifnum\forestOve{#1}{@parent}=0
1387 \PackageError{forest}{get next leaf: this is the last leaf}{}%
1388 \else
1389 \escapeif{\expandnumberarg\forest@node@Nextleafid{\forestOve{#1}{@parent}}}%

```

```

1390 \fi
1391 }
1392 \def\forest@node@nextleafid@Gdown#1{%
1393 \ifnum\forestOve{#1}{@first}=0
1394 #1%
1395 \else
1396 \@escapeif{\expandnumberarg\forest@node@nextleafid@Gdown{\forestOve{#1}{@first}}}%
1397 \fi
1398 }
1399
1400
1401
1402 \def\forest@node@linearnextid{%
1403 \ifnum\forestove{@first}=0
1404 \expandafter\forest@node@linearnextnotdescendantid
1405 \else
1406 \forestove{@first}%
1407 \fi
1408 }
1409 \def\forest@node@linearnextnotdescendantid{%
1410 \expandnumberarg\forest@node@Linearnextnotdescendantid{\forest@cn}%
1411 }
1412 \def\forest@node@Linearnextnotdescendantid#1{%
1413 \ifnum\forestOve{#1}{@next}=0
1414 \ifnum\forestOve{#1}{@parent}=0
1415 0%
1416 \else
1417 \@escapeif{\expandnumberarg\forest@node@Linearnextnotdescendantid{\forestOve{#1}{@parent}}}%
1418 \fi
1419 \else
1420 \forestOve{#1}{@next}%
1421 \fi
1422 }
1423
1424
1425 \def\forest@node@linearpreviousid{%
1426 \ifnum\forestove{@previous}=0
1427 \forestove{@parent}%
1428 \else
1429 \forest@node@previousleafid
1430 \fi
1431 }
1432 \def\forest@ifancestorof#1{% is the current node an ancestor of #1? Yes: #2, no: #3
1433 \expandnumberarg\forest@ifancestorof@{\forestOve{#1}{@parent}}%
1434 }
1435 \def\forest@ifancestorof@#1#2#3{%
1436 \ifnum#1=0
1437 \def\forest@ifancestorof@next{\@secondoftwo}%
1438 \else
1439 \ifnum\forest@cn=#1
1440 \def\forest@ifancestorof@next{\@firstoftwo}%
1441 \else
1442 \def\forest@ifancestorof@next{\expandnumberarg\forest@ifancestorof@{\forestOve{#1}{@parent}}}%
1443 \fi
1444 \fi
1445 \forest@ifancestorof@next{#2}{#3}%
1446 }

```

6.3 Node options

6.3.1 Option-declaration mechanism

Common code for declaring options.

```
1447 \def\forest@declarehandler#1#2#3{% #1=handler for specific type,#2=option name,#3=default value
1448   \pgfkeyssetvalue{/forest/#2}{#3}%
1449   \appto\forest@node@init{\forest@init{#2}}%
1450   \pgfkeyssetvalue{/forest/#2/node@or@reg}{\forest@cn}%
1451   \forest@convert@others@to@underscores{#2}\forest@pgfmathoptionname
1452   \edef\forest@marshal{%
1453     \noexpand#1{/forest/#2}{/forest}{#2}{\forest@pgfmathoptionname}%
1454   }\forest@marshal
1455 }
1456 \def\forest@def@with@pgfeov#1#2{% \pgfeov mustn't occur in the arg of the .code handler!!!
1457   \long\def#1##1\pgfeov{#2}%
1458 }
```

Option-declaration handlers.

```
1459 \def\forest@declaretokens@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmathname
1460   \forest@declaretokens@handler@A{#1}{#2}{#3}{#4}{}%
1461 }
1462 \def\forest@declarekeylist@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmathname
1463   \forest@declaretokens@handler@A{#1}{#2}{#3}{#4}{,}%
1464   \forest@copycommandkey{#1}{#1'}%
1465   \pgfkeyssetvalue{#1'/option@name}{#3}%
1466   \forest@copycommandkey{#1+}{#1}%
1467   \pgfkeysalso{#1-/.code={%
1468     \forest@for@node{\forest@setter@node}{%
1469       \forest@node@removekeysfromkeylist{##1}{#3}%
1470     }}%
1471   \pgfkeyssetvalue{#1-/.option@name}{#3}%
1472 }
1473 \def\forest@declaretokens@handler@A#1#2#3#4#5{% #1=key,#2=path,#3=name,#4=pgfmathname,#5=infix
1474   \pgfkeysalso{%
1475     #1/.code={\forest@set{\forest@setter@node}{#3}{##1}},
1476     #1+/.code={\forest@appto{\forest@setter@node}{#3}{#5##1}},
1477     #2/+#3/.code={\forest@preto{\forest@setter@node}{#3}{##1#5}},
1478     #2/if #3/.code n args={3}{%
1479       \forest@toget{#3}\forest@temp@option@value
1480       \edef\forest@temp@compared@value{\unexpanded{##1}}%
1481       \ifx\forest@temp@option@value\forest@temp@compared@value
1482         \pgfkeysalso{##2}%
1483       \else
1484         \pgfkeysalso{##3}%
1485       \fi
1486     },
1487     #2/if in #3/.code n args={3}{%
1488       \forest@toget{#3}\forest@temp@option@value
1489       \edef\forest@temp@compared@value{\unexpanded{##1}}%
1490       \expandafter\expandafter\expandafter\pgfutil@in@\expandafter\expandafter\expandafter{\expandafter\forest@temp@option@value}
1491       \ifpgfutil@in@
1492         \pgfkeysalso{##2}%
1493       \else
1494         \pgfkeysalso{##3}%
1495       \fi
1496     },
1497     #2/where #3/.style n args={3}{for tree={#2/if #3={##1}{##2}{##3}}},
1498     #2/where in #3/.style n args={3}{for tree={#2/if in #3={##1}{##2}{##3}}}
1499   }%
1500   \pgfkeyssetvalue{#1/option@name}{#3}%
1501   \pgfkeyssetvalue{#1+/.option@name}{#3}%

```

```

1502 \pgfkeyssetvalue{#2/+ #3/option@name}{#3}%
1503 \pgfmathdeclarefunction{#4}{1}{\forest@pgfmathhelper@attribute@toks{##1}{#3}}%
1504 }
1505 \def\forest@declareautowrappedtoks@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmathname,#5=infix
1506 \forest@declaretoks@handler{#1}{#2}{#3}{#4}%
1507 \forest@copycommandkey{#1}{#1'}%
1508 \pgfkeysalso{#1/.style={#1'/.wrap value={##1}}}%
1509 \pgfkeyssetvalue{#1'/option@name}{#3}%
1510 \forest@copycommandkey{#1+}{#1+'}%
1511 \pgfkeysalso{#1+/.style={#1+'/.wrap value={##1}}}%
1512 \pgfkeyssetvalue{#1+'/option@name}{#3}%
1513 \forest@copycommandkey{#2/+ #3}{#2/+ #3'}%
1514 \pgfkeysalso{#2/+ #3/.style={#2/+ #3'/.wrap value={##1}}}%
1515 \pgfkeyssetvalue{#2/+ #3'/option@name}{#3}%
1516 }
1517 \def\forest@declarereadonlydimen@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmathname
1518 \pgfkeysalso{%
1519 #2/if #3/.code n args={3}{%
1520 \forest@toget{#3}\forest@temp@option@value
1521 \ifdim\forest@temp@option@value=##1\relax
1522 \pgfkeysalso{##2}%
1523 \else
1524 \pgfkeysalso{##3}%
1525 \fi
1526 },
1527 #2/where #3/.style n args={3}{for tree={#2/if #3={##1}{##2}{##3}}},
1528 }%
1529 \pgfmathdeclarefunction{#4}{1}{\forest@pgfmathhelper@attribute@dimen{##1}{#3}}%
1530 }
1531 \def\forest@declaredimen@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmathname
1532 \forest@declarereadonlydimen@handler{#1}{#2}{#3}{#4}%
1533 \pgfkeysalso{%
1534 #1/.code={%
1535 \pgfmathsetlengthmacro\forest@temp{##1}%
1536 \forest@0let{\forest@setter@node}{#3}\forest@temp
1537 },
1538 #1+/.code={%
1539 \pgfmathsetlengthmacro\forest@temp{##1}%
1540 \pgfutil@tempdima=\forest@temp{#3}
1541 \advance\pgfutil@tempdima\forest@temp\relax
1542 \forest@0set{\forest@setter@node}{#3}{\the\pgfutil@tempdima}%
1543 },
1544 #1-/.code={%
1545 \pgfmathsetlengthmacro\forest@temp{##1}%
1546 \pgfutil@tempdima=\forest@temp{#3}
1547 \advance\pgfutil@tempdima-\forest@temp\relax
1548 \forest@0set{\forest@setter@node}{#3}{\the\pgfutil@tempdima}%
1549 },
1550 #1*/.style={%
1551 #1={#4()*(##1)}%
1552 },
1553 #1:/ .style={%
1554 #1={#4()/ (##1)}%
1555 },
1556 #1'/.code={%
1557 \pgfutil@tempdima=##1\relax
1558 \forest@0set{\forest@setter@node}{#3}{\the\pgfutil@tempdima}%
1559 },
1560 #1' +/.code={%
1561 \pgfutil@tempdima=\forest@temp{#3}\relax
1562 \advance\pgfutil@tempdima##1\relax

```

```

1563 \forestOeset{\forest@setter@node}{#3}{\the\pgfutil@tempdima}%
1564 },
1565 #1'-.code={%
1566 \pgfutil@tempdima=\forestove{#3}\relax
1567 \advance\pgfutil@tempdima-##1\relax
1568 \forestOeset{\forest@setter@node}{#3}{\the\pgfutil@tempdima}%
1569 },
1570 #1'*/.style={%
1571 \pgfutil@tempdima=\forestove{#3}\relax
1572 \multiply\pgfutil@tempdima##1\relax
1573 \forestOeset{\forest@setter@node}{#3}{\the\pgfutil@tempdima}%
1574 },
1575 #1':/.style={%
1576 \pgfutil@tempdima=\forestove{#3}\relax
1577 \divide\pgfutil@tempdima##1\relax
1578 \forestOeset{\forest@setter@node}{#3}{\the\pgfutil@tempdima}%
1579 },
1580 }%
1581 \pgfkeyssetvalue{#1/option@name}{#3}%
1582 \pgfkeyssetvalue{#1+/option@name}{#3}%
1583 \pgfkeyssetvalue{#1-/option@name}{#3}%
1584 \pgfkeyssetvalue{#1*/option@name}{#3}%
1585 \pgfkeyssetvalue{#1:/option@name}{#3}%
1586 \pgfkeyssetvalue{#1'/option@name}{#3}%
1587 \pgfkeyssetvalue{#1'+/option@name}{#3}%
1588 \pgfkeyssetvalue{#1'-/option@name}{#3}%
1589 \pgfkeyssetvalue{#1'*/option@name}{#3}%
1590 \pgfkeyssetvalue{#1':/option@name}{#3}%
1591 }
1592 \def\forest@declarereadonlycount@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmathname
1593 \pgfkeysalso{
1594 #2/if #3/.code n args={3}{%
1595 \forestotget{#3}\forest@temp@option@value
1596 \ifnum\forest@temp@option@value=##1\relax
1597 \pgfkeysalso{##2}%
1598 \else
1599 \pgfkeysalso{##3}%
1600 \fi
1601 },
1602 #2/where #3/.style n args={3}{for tree={#2/if #3={##1}{##2}{##3}}},
1603 }%
1604 \pgfmathdeclarefunction{#4}{1}{\forest@pgfmathhelper@attribute@count{##1}{#3}}%
1605 }
1606 \def\forest@declarecount@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmathname
1607 \forest@declarereadonlycount@handler{#1}{#2}{#3}{#4}%
1608 \pgfkeysalso{
1609 #1/.code={%
1610 \pgfmathtruncatemacro\forest@temp{##1}%
1611 \forestOlet{\forest@setter@node}{#3}\forest@temp
1612 },
1613 #1+/.code={%
1614 \pgfmathtruncatemacro\forest@temp{##1}%
1615 \c@pgf@counta=\forestove{#3}\relax
1616 \advance\c@pgf@counta\forest@temp\relax
1617 \forestOeset{\forest@setter@node}{#3}{\the\c@pgf@counta}%
1618 },
1619 #1-/.code={%
1620 \pgfmathtruncatemacro\forest@temp{##1}%
1621 \c@pgf@counta=\forestove{#3}\relax
1622 \advance\c@pgf@counta-\forest@temp\relax
1623 \forestOeset{\forest@setter@node}{#3}{\the\c@pgf@counta}%

```



```

1624 },
1625 #1*/.code={%
1626   \pgfmathtruncatemacro\forest@temp{##1}%
1627   \c@pgf@counta=\forestove{#3}\relax
1628   \multiply\c@pgf@counta\forest@temp\relax
1629   \forestOset{\forest@setter@node}{#3}{\the\c@pgf@counta}%
1630 },
1631 #1:/.code={%
1632   \pgfmathtruncatemacro\forest@temp{##1}%
1633   \c@pgf@counta=\forestove{#3}\relax
1634   \divide\c@pgf@counta\forest@temp\relax
1635   \forestOset{\forest@setter@node}{#3}{\the\c@pgf@counta}%
1636 },
1637 #1'/.code={%
1638   \c@pgf@counta=##1\relax
1639   \forestOset{\forest@setter@node}{#3}{\the\c@pgf@counta}%
1640 },
1641 #1'+/.code={%
1642   \c@pgf@counta=\forestove{#3}\relax
1643   \advance\c@pgf@counta##1\relax
1644   \forestOset{\forest@setter@node}{#3}{\the\c@pgf@counta}%
1645 },
1646 #1'-/.code={%
1647   \c@pgf@counta=\forestove{#3}\relax
1648   \advance\c@pgf@counta-##1\relax
1649   \forestOset{\forest@setter@node}{#3}{\the\c@pgf@counta}%
1650 },
1651 #1'*/.style={%
1652   \c@pgf@counta=\forestove{#3}\relax
1653   \multiply\c@pgf@counta##1\relax
1654   \forestOset{\forest@setter@node}{#3}{\the\c@pgf@counta}%
1655 },
1656 #1':/.style={%
1657   \c@pgf@counta=\forestove{#3}\relax
1658   \divide\c@pgf@counta##1\relax
1659   \forestOset{\forest@setter@node}{#3}{\the\c@pgf@counta}%
1660 },
1661 }%
1662 \pgfkeyssetvalue{#1/option@name}{#3}%
1663 \pgfkeyssetvalue{#1+/option@name}{#3}%
1664 \pgfkeyssetvalue{#1-/option@name}{#3}%
1665 \pgfkeyssetvalue{#1*/option@name}{#3}%
1666 \pgfkeyssetvalue{#1:/option@name}{#3}%
1667 \pgfkeyssetvalue{#1'/option@name}{#3}%
1668 \pgfkeyssetvalue{#1'+/option@name}{#3}%
1669 \pgfkeyssetvalue{#1'-/option@name}{#3}%
1670 \pgfkeyssetvalue{#1'* /option@name}{#3}%
1671 \pgfkeyssetvalue{#1':/option@name}{#3}%
1672 }
1673 \def\forest@declareboolean@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmathname
1674   \pgfkeysalso{%
1675     #1/.code={%
1676       \ifstrequal{##1}{1}{%
1677         \forestOset{\forest@setter@node}{#3}{1}%
1678       }{%
1679         \ifstrequal{##1}{0}{%
1680           \forestOset{\forest@setter@node}{#3}{0}%
1681         }{%
1682           \pgfmathifthenelse{##1}{1}{0}%
1683           \forestOlet{\forest@setter@node}{#3}\pgfmathresult
1684         }%
1685       }%
1686     }%
1687   }%

```

```

1685     }%
1686 },
1687 #1/.default=1,
1688 #2/not #3/.code={\forestOset{\forest@setter@node}{#3}{0}},
1689 #2/if #3/.code 2 args={%
1690     \forestoget{#3}\forest@temp@option@value
1691     \ifnum\forest@temp@option@value=1
1692         \pgfkeysalso{##1}%
1693     \else
1694         \pgfkeysalso{##2}%
1695     \fi
1696 },
1697 #2/where #3/.style 2 args={for tree={#2/if #3={##1}{##2}}}
1698 }%
1699 \pgfkeyssetvalue{#1/option@name}{#3}%
1700 \pgfmathdeclarefunction{#4}{1}{\forest@pgfmathhelper@attribute@count{##1}{#3}}%
1701 }
1702 \forestset{
1703     declare toks/.code 2 args={%
1704         \forest@declarehandler\forest@declaretoks@handler{#1}{#2}%
1705     },
1706     declare autowrapped toks/.code 2 args={%
1707         \forest@declarehandler\forest@declareautowrappedtoks@handler{#1}{#2}%
1708     },
1709     declare keylist/.code 2 args={%
1710         \forest@declarehandler\forest@declarekeylist@handler{#1}{#2}%
1711     },
1712     declare readonly dimen/.code={%
1713         \forest@declarehandler\forest@declarereadonlydimen@handler{#1}{}%
1714     },
1715     declare dimen/.code 2 args={%
1716         \forest@declarehandler\forest@declaredimen@handler{#1}{#2}%
1717     },
1718     declare readonly count/.code={%
1719         \forest@declarehandler\forest@declarereadonlycount@handler{#1}{}%
1720     },
1721     declare count/.code 2 args={%
1722         \forest@declarehandler\forest@declarecount@handler{#1}{#2}%
1723     },
1724     declare boolean/.code 2 args={%
1725         \forest@declarehandler\forest@declareboolean@handler{#1}{#2}%
1726     },
1727     /handlers/.restore default value/.code={%
1728         \edef\forest@handlers@currentpath{\pgfkeyscurrentpath}%
1729         \pgfkeysgetvalue{\pgfkeyscurrentpath/option@name}\forest@currentoptionname
1730         \pgfkeysgetvalue{/forest/\forest@currentoptionname}\forest@temp
1731         \expandafter\pgfkeysalso\expandafter{\forest@handlers@currentpath/.expand once=\forest@temp}%
1732     },
1733     /handlers/.pgfmath/.code={%
1734         \pgfmathparse{#1}%
1735         \pgfkeysalso{\pgfkeyscurrentpath/.expand once=\pgfmathresult}%
1736     },
1737     /handlers/.wrap value/.code={%
1738         \edef\forest@handlers@wrap@currentpath{\pgfkeyscurrentpath}%
1739         \pgfkeysgetvalue{\forest@handlers@wrap@currentpath/option@name}\forest@currentoptionname
1740         \forestOget{\pgfkeysvalueof{/forest/\forest@currentoptionname/node@or@reg}}{\forest@currentoptionname}\forest@temp
1741         \forest@def@with@pgfeov\forest@wrap@code{#1}%
1742         \expandafter\edef\expandafter\forest@wrapped@value\expandafter{\expandafter\expandonce\expandafter{\expandafter\forest@temp}}
1743         \pgfkeysalso{\forest@handlers@wrap@currentpath/.expand once=\forest@wrapped@value}%
1744     },
1745     /handlers/.option/.code={%

```

```

1746 \edef\forest@marshal{%
1747 \noexpand\pgfkeysalso{\pgfkeyscurrentpath={\forestoption{#1}}}%
1748 }\forest@marshal
1749 },
1750 /handlers/.register/.code={%
1751 \edef\forest@marshal{%
1752 \noexpand\pgfkeysalso{\pgfkeyscurrentpath={\forestregister{#1}}}%
1753 }\forest@marshal
1754 },
1755 /handlers/.wrap pgfmath arg/.code 2 args={%
1756 \pgfmathparse{#2}\let\forest@wrap@arg@i\pgfmathresult
1757 \edef\forest@wrap@args{{\expandonce\forest@wrap@arg@i}}%
1758 \def\forest@wrap@code##1{#1}%
1759 \expandafter\expandafter\expandafter\forest@temp@toks\expandafter\expandafter\expandafter{\expandafter\fo
1760 \pgfkeysalso{\pgfkeyscurrentpath/.expand once=\the\forest@temp@toks}%
1761 },
1762 /handlers/.wrap 2 pgfmath args/.code n args={3}{%
1763 \pgfmathparse{#2}\let\forest@wrap@arg@i\pgfmathresult
1764 \pgfmathparse{#3}\let\forest@wrap@arg@ii\pgfmathresult
1765 \edef\forest@wrap@args{{\expandonce\forest@wrap@arg@i}{\expandonce\forest@wrap@arg@ii}}%
1766 \def\forest@wrap@code##1##2{#1}%
1767 \expandafter\expandafter\expandafter\def\expandafter\expandafter\expandafter\forest@wrapped\expandafter\fo
1768 \pgfkeysalso{\pgfkeyscurrentpath/.expand once=\forest@wrapped}%
1769 },
1770 /handlers/.wrap 3 pgfmath args/.code n args={4}{%
1771 \forest@wrap@n@pgfmath@args{#2}{#3}{#4}{-}{-}{-}{-}{3}%
1772 \forest@wrap@n@pgfmath@do{#1}{3}},
1773 /handlers/.wrap 4 pgfmath args/.code n args={5}{%
1774 \forest@wrap@n@pgfmath@args{#2}{#3}{#4}{#5}{-}{-}{-}{-}{4}%
1775 \forest@wrap@n@pgfmath@do{#1}{4}},
1776 /handlers/.wrap 5 pgfmath args/.code n args={6}{%
1777 \forest@wrap@n@pgfmath@args{#2}{#3}{#4}{#5}{#6}{-}{-}{-}{5}%
1778 \forest@wrap@n@pgfmath@do{#1}{5}},
1779 /handlers/.wrap 6 pgfmath args/.code n args={7}{%
1780 \forest@wrap@n@pgfmath@args{#2}{#3}{#4}{#5}{#6}{#7}{-}{-}{6}%
1781 \forest@wrap@n@pgfmath@do{#1}{6}},
1782 /handlers/.wrap 7 pgfmath args/.code n args={8}{%
1783 \forest@wrap@n@pgfmath@args{#2}{#3}{#4}{#5}{#6}{#7}{#8}{-}{7}%
1784 \forest@wrap@n@pgfmath@do{#1}{7}},
1785 /handlers/.wrap 8 pgfmath args/.code n args={9}{%
1786 \forest@wrap@n@pgfmath@args{#2}{#3}{#4}{#5}{#6}{#7}{#8}{#9}{8}%
1787 \forest@wrap@n@pgfmath@do{#1}{8}},
1788 /handlers/.process args/.code={%
1789 \forest@processargs#1\forest@eov\forest@END
1790 },
1791 }
1792 \def\forest@wrap@n@pgfmath@args#1#2#3#4#5#6#7#8#9{%
1793 \pgfmathparse{#1}\let\forest@wrap@arg@i\pgfmathresult
1794 \ifnum#9>1 \pgfmathparse{#2}\let\forest@wrap@arg@ii\pgfmathresult\fi
1795 \ifnum#9>2 \pgfmathparse{#3}\let\forest@wrap@arg@iii\pgfmathresult\fi
1796 \ifnum#9>3 \pgfmathparse{#4}\let\forest@wrap@arg@iv\pgfmathresult\fi
1797 \ifnum#9>4 \pgfmathparse{#5}\let\forest@wrap@arg@v\pgfmathresult\fi
1798 \ifnum#9>5 \pgfmathparse{#6}\let\forest@wrap@arg@vi\pgfmathresult\fi
1799 \ifnum#9>6 \pgfmathparse{#7}\let\forest@wrap@arg@vii\pgfmathresult\fi
1800 \ifnum#9>7 \pgfmathparse{#8}\let\forest@wrap@arg@viii\pgfmathresult\fi
1801 \edef\forest@wrap@args{%
1802 {\expandonce\forest@wrap@arg@i}
1803 \ifnum#9>1 {\expandonce\forest@wrap@arg@ii}\fi
1804 \ifnum#9>2 {\expandonce\forest@wrap@arg@iii}\fi
1805 \ifnum#9>3 {\expandonce\forest@wrap@arg@iv}\fi
1806 \ifnum#9>4 {\expandonce\forest@wrap@arg@v}\fi

```

```

1807 \ifnum#9>5 {\expandonce\forest@wrap@arg@vi}\fi
1808 \ifnum#9>6 {\expandonce\forest@wrap@arg@vii}\fi
1809 \ifnum#9>7 {\expandonce\forest@wrap@arg@viii}\fi
1810 }%
1811 }
1812 \def\forest@wrap@n@pgfmath@do#1#2{%
1813 \ifcase#2\relax
1814 \or\def\forest@wrap@code##1{#1}%
1815 \or\def\forest@wrap@code##1##2{#1}%
1816 \or\def\forest@wrap@code##1##2##3{#1}%
1817 \or\def\forest@wrap@code##1##2##3##4{#1}%
1818 \or\def\forest@wrap@code##1##2##3##4##5{#1}%
1819 \or\def\forest@wrap@code##1##2##3##4##5##6{#1}%
1820 \or\def\forest@wrap@code##1##2##3##4##5##6##7{#1}%
1821 \or\def\forest@wrap@code##1##2##3##4##5##6##7##8{#1}%
1822 \fi
1823 \expandafter\expandafter\expandafter\def\expandafter\expandafter\expandafter\forest@wrapped\expandafter\exp
1824 \pgfkeysalso{\pgfkeyscurrentpath/.expand once=\forest@wrapped}%
1825 }
1826 \newtoks\forest@processargs@result
1827 \newtoks\forest@processargs@current
1828 \newif\ifforest@processargs@append
1829 \def\forest@eov{\forest@eov}
1830 \def\forest@processargs#1#2\forest@END{% #1 = processing instructions, #2 = args
1831 \forest@processargs@result{}%
1832 \forest@processargs@current{}%
1833 \forest@processargs@appendfalse
1834 \forest@processargs@getins#1.\forest@END#2\forest@END
1835 }
1836 \def\forest@processargs@maybeappend{%
1837 \ifforest@processargs@append
1838 \eapptotoks\forest@processargs@result{{\the\forest@processargs@current}}%
1839 \forest@processargs@current{}%
1840 \fi
1841 }
1842 \def\forest@processargs@getins#1#2\forest@END#3\forest@END{% #1 = first instruction, #2 = rest of instruction
1843 \csname forest@processargs@ins@#1\endcsname#2\forest@END#3\forest@END
1844 }
1845 \csdef{forest@processargs@ins@.}\forest@END#1\forest@END{%
1846 \forest@processargs@maybeappend
1847 \forest@processargs@appremainingargs#1\forest@END
1848 }
1849 \def\forest@processargs@appremainingargs#1#2\forest@END{%
1850 \edef\forest@temp{\unexpanded{#1}}%
1851 \ifx\forest@temp\forest@eov
1852 \let\forest@processargs@next\forest@processargs@done
1853 \else
1854 \apptotoks\forest@processargs@result{{#1}}%
1855 \let\forest@processargs@next\forest@processargs@appremainingargs
1856 \fi
1857 \forest@processargs@next#2\forest@END
1858 }
1859 \def\forest@processargs@done#1\forest@END{%
1860 \pgfkeysalso{\pgfkeyscurrentpath/.expanded=\the\forest@processargs@result}%
1861 }
1862 \csdef{forest@processargs@ins@.}#1\forest@END#2#3\forest@END{% no processing
1863 % #1 = rest of ins, #2 = first arg, #3 = rest of args
1864 \forest@processargs@maybeappend
1865 \forest@processargs@current{#2}%
1866 \forest@processargs@appendtrue
1867 \forest@processargs@getins#1\forest@END#3\forest@END

```

```

1868 }
1869 \csdef{forest@processargs@ins@x}#1\forest@END#2#3\forest@END{% expand
1870 % #1 = rest of ins, #2 = first arg, #3 = rest of args
1871 \forest@processargs@maybeappend
1872 \etotoks\forest@processargs@current{#2}%
1873 \forest@processargs@appendtrue
1874 \forest@processargs@getins#1\forest@END#3\forest@END
1875 }
1876 \csdef{forest@processargs@ins@o}#1\forest@END#2#3\forest@END{% expand once
1877 % #1 = rest of ins, #2 = first arg, #3 = rest of args
1878 \forest@processargs@maybeappend
1879 \expandafter\forest@processargs@current\expandafter{#2}%
1880 \forest@processargs@appendtrue
1881 \forest@processargs@getins#1\forest@END#3\forest@END
1882 }
1883 \csdef{forest@processargs@ins@O}#1\forest@END#2#3\forest@END{% option
1884 % #1 = rest of ins, #2 = first arg, #3 = rest of args
1885 \forest@processargs@maybeappend
1886 \etotoks\forest@processargs@current{\forestoption{#2}}%
1887 \forest@processargs@appendtrue
1888 \forest@processargs@getins#1\forest@END#3\forest@END
1889 }
1890 \csdef{forest@processargs@ins@R}#1\forest@END#2#3\forest@END{% register
1891 % #1 = rest of ins, #2 = first arg, #3 = rest of args
1892 \forest@processargs@maybeappend
1893 \etotoks\forest@processargs@current{\forestregister{#2}}%
1894 \forest@processargs@appendtrue
1895 \forest@processargs@getins#1\forest@END#3\forest@END
1896 }
1897 \csdef{forest@processargs@ins@P}#1\forest@END#2#3\forest@END{% pgfmath expression
1898 % #1 = rest of ins, #2 = first arg, #3 = rest of args
1899 \forest@processargs@maybeappend
1900 \pgfmathparse{#2}%
1901 \expandafter\forest@processargs@current\expandafter{\pgfmathresult}%
1902 \forest@processargs@appendtrue
1903 \forest@processargs@getins#1\forest@END#3\forest@END
1904 }
1905 \csdef{forest@processargs@ins@+}#1\forest@END#2\forest@END{% join processors
1906 \forest@processargs@appendfalse
1907 \edef\forest@marshal{%
1908 \unexpanded{\forest@processargs@getins#1\forest@END}{\the\forest@processargs@current}\unexpanded{#2\forest@processargs@getins#1\forest@END}
1909 }\forest@marshal
1910 }
1911 \csdef{forest@processargs@ins@r}#1\forest@END#2#3\forest@END{% reverse keylist
1912 % #1 = rest of ins, #2 = first arg, #3 = rest of args
1913 \forest@processargs@maybeappend
1914 \forest@processargs@current{}%
1915 \pgfqkeys{/forest}{split={#2}{,}{reverse@keylist}}%
1916 \forest@processargs@appendtrue
1917 \forest@processargs@getins#1\forest@END#3\forest@END
1918 }
1919 \forestset{%
1920 reverse@keylist/.code={%
1921 \epretotoks\forest@processargs@current{#1,}%
1922 },
1923 }

```

6.3.2 Registers

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

```

1924 \def\forest@pgfmathhelper@register@toks#1#2{% #1 is discarded: it is present only for analogy with options
1925   \forestrget{#2}\pgfmathresult
1926 }
1927 \def\forest@pgfmathhelper@register@dimen#1#2{%
1928   \forestrget{#2}\forest@temp
1929   \pgfmathparse{+\forest@temp}%
1930 }
1931 \def\forest@pgfmathhelper@register@count#1#2{%
1932   \forestrget{#2}\forest@temp
1933   \pgfmathtruncatemacro\pgfmathresult{\forest@temp}%
1934 }
1935 \def\forest@declareregisterhandler#1#2{%#1=handler for specific type,#2=option name
1936   \pgfkeyssetvalue{/forest/#2/node@or@reg}{register}%
1937   \forest@convert@others@to@underscores{#2}\forest@pgfmathoptionname
1938   \edef\forest@marshal{%
1939     \noexpand#1{/forest/#2}/{/forest}{#2}{\forest@pgfmathoptionname}%
1940   }\forest@marshal
1941 }
1942 \def\forest@declaretoksregister@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmathname
1943   \forest@declaretoksregister@handler@A{#1}{#2}{#3}{#4}{}%
1944 }
1945 \def\forest@declarekeylistregister@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmathname
1946   \forest@declaretoksregister@handler@A{#1}{#2}{#3}{#4}{,}%
1947   \forest@copycommandkey{#1}{#1'}%
1948   \pgfkeyssetvalue{#1'/option@name}{#3}%
1949   \forest@copycommandkey{#1+}{#1}%
1950   \pgfkeysalso{#1-/.code={%
1951     \forest@fornode{register}{%
1952       \forest@node@removekeysfromkeylist{##1}{#3}%
1953     }}%
1954   \pgfkeyssetvalue{#1- /option@name}{#3}%
1955 }
1956 \def\forest@declaretoksregister@handler@A#1#2#3#4#5{% #1=key,#2=path,#3=name,#4=pgfmathname,#5=infix
1957   \pgfkeysalso{%
1958     #1/.code={\forestrset{#3}{##1}},
1959     #1+/.code={\forestrappto{#3}{#5##1}},
1960     #2/+#3/.code={\forestrpreto{#3}{##1#5}},
1961     #2/if #3/.code n args={3}{%
1962       \forestrget{#3}\forest@temp@option@value
1963       \edef\forest@temp@compared@value{\unexpanded{##1}}%
1964       \ifx\forest@temp@option@value\forest@temp@compared@value
1965         \pgfkeysalso{##2}%
1966       \else
1967         \pgfkeysalso{##3}%
1968       \fi
1969     },
1970     #2/if in #3/.code n args={3}{%
1971       \forestrget{#3}\forest@temp@option@value
1972       \edef\forest@temp@compared@value{\unexpanded{##1}}%
1973       \expandafter\expandafter\expandafter\pgfutil@in@\expandafter\expandafter\expandafter{\expandafter\forest@temp@option@value\forest@temp@compared@value}
1974       \ifpgfutil@in@
1975         \pgfkeysalso{##2}%
1976       \else
1977         \pgfkeysalso{##3}%
1978       \fi
1979     },
1980   }%
1981   \pgfkeyssetvalue{#1/option@name}{#3}%
1982   \pgfkeyssetvalue{#1+ /option@name}{#3}%
1983   \pgfkeyssetvalue{#2/+#3/option@name}{#3}%
1984   \pgfmathdeclarefunction{#4}{1}{\forest@pgfmathhelper@register@toks{##1}{#3}}%

```

```

1985 }
1986 \def\forest@declareautowrappedtoksregister@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmname,#5=infix
1987 \forest@declaretoksregister@handler{#1}{#2}{#3}{#4}%
1988 \forest@copycommandkey{#1}{#1'}%
1989 \pgfkeysalso{#1/.style={#1'/.wrap value={##1}}}%
1990 \pgfkeyssetvalue{#1'/option@name}{#3}%
1991 \forest@copycommandkey{#1+}{#1+'}%
1992 \pgfkeysalso{#1+/.style={#1+'/.wrap value={##1}}}%
1993 \pgfkeyssetvalue{#1+'/option@name}{#3}%
1994 \forest@copycommandkey{#2/+}{#2/+}{#2/+}{#3}%
1995 \pgfkeysalso{#2/+}{#2/+}{#2/+}{#3}%
1996 \pgfkeyssetvalue{#2/+}{#3'/option@name}{#3}%
1997 }
1998 \def\forest@declarereadonlydimenregister@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmname
1999 \pgfkeysalso{%
2000 #2/if #3/.code n args={3}{%
2001 \forest@temp@option@value
2002 \ifdim\forest@temp@option@value=##1\relax
2003 \pgfkeysalso{##2}%
2004 \else
2005 \pgfkeysalso{##3}%
2006 \fi
2007 },
2008 }%
2009 \pgfmnamedeclarefunction{#4}{1}{\forest@pgfmnamehelper@register@dimen{##1}{#3}}%
2010 }
2011 \def\forest@declaredimenregister@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmname
2012 \forest@declarereadonlydimenregister@handler{#1}{#2}{#3}{#4}%
2013 \pgfkeysalso{%
2014 #1/.code={%
2015 \pgfmname@setlengthmacro\forest@temp{##1}%
2016 \forest@let{#3}\forest@temp
2017 },
2018 #1+/.code={%
2019 \pgfmname@setlengthmacro\forest@temp{##1}%
2020 \pgfutil@tempdima=\forest@temp{#3}
2021 \advance\pgfutil@tempdima\forest@temp\relax
2022 \forest@reset{#3}{\the\pgfutil@tempdima}%
2023 },
2024 #1-/.code={%
2025 \pgfmname@setlengthmacro\forest@temp{##1}%
2026 \pgfutil@tempdima=\forest@temp{#3}
2027 \advance\pgfutil@tempdima-\forest@temp\relax
2028 \forest@reset{#3}{\the\pgfutil@tempdima}%
2029 },
2030 #1*/.style={%
2031 #1={#4()}*(##1)}%
2032 },
2033 #1:/ .style={%
2034 #1={#4()/(##1)}%
2035 },
2036 #1'/.code={%
2037 \pgfutil@tempdima=##1\relax
2038 \forest@reset{#3}{\the\pgfutil@tempdima}%
2039 },
2040 #1'+/.code={%
2041 \pgfutil@tempdima=\forest@temp{#3}\relax
2042 \advance\pgfutil@tempdima##1\relax
2043 \forest@reset{#3}{\the\pgfutil@tempdima}%
2044 },
2045 #1'-/.code={%

```

```

2046     \pgfutil@tempdima=\forestrve{#3}\relax
2047     \advance\pgfutil@tempdima-##1\relax
2048     \forestreset{#3}{\the\pgfutil@tempdima}%
2049 },
2050 #1'*/.style={%
2051     \pgfutil@tempdima=\forestrve{#3}\relax
2052     \multiply\pgfutil@tempdima##1\relax
2053     \forestreset{#3}{\the\pgfutil@tempdima}%
2054 },
2055 #1':/.style={%
2056     \pgfutil@tempdima=\forestrve{#3}\relax
2057     \divide\pgfutil@tempdima##1\relax
2058     \forestreset{#3}{\the\pgfutil@tempdima}%
2059 },
2060 }%
2061 \pgfkeyssetvalue{#1/option@name}{#3}%
2062 \pgfkeyssetvalue{#1+/option@name}{#3}%
2063 \pgfkeyssetvalue{#1-/option@name}{#3}%
2064 \pgfkeyssetvalue{#1*/option@name}{#3}%
2065 \pgfkeyssetvalue{#1:/option@name}{#3}%
2066 \pgfkeyssetvalue{#1'/option@name}{#3}%
2067 \pgfkeyssetvalue{#1'+/option@name}{#3}%
2068 \pgfkeyssetvalue{#1'-/option@name}{#3}%
2069 \pgfkeyssetvalue{#1'*-/option@name}{#3}%
2070 \pgfkeyssetvalue{#1':/option@name}{#3}%
2071 }
2072 \def\forest@declarereadonlycountregister@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmathname
2073     \pgfkeysalso{
2074         #2/if #3/.code n args={3}{%
2075             \forestrget{#3}\forest@temp@option@value
2076             \ifnum\forest@temp@option@value=##1\relax
2077                 \pgfkeysalso{##2}%
2078             \else
2079                 \pgfkeysalso{##3}%
2080             \fi
2081         },
2082     }%
2083     \pgfmathdeclarefunction{#4}{1}{\forest@pgfmathhelper@register@count{##1}{#3}}%
2084 }
2085 \def\forest@declarecountregister@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmathname
2086     \forest@declarereadonlycountregister@handler{#1}{#2}{#3}{#4}%
2087     \pgfkeysalso{
2088         #1/.code={%
2089             \pgfmathtruncatemacro\forest@temp{##1}%
2090             \forestrllet{#3}\forest@temp
2091         },
2092         #1+/.code={%
2093             \pgfmathtruncatemacro\forest@temp{##1}%
2094             \c@pgf@counta=\forestrve{#3}\relax
2095             \advance\c@pgf@counta\forest@temp\relax
2096             \forestreset{#3}{\the\c@pgf@counta}%
2097         },
2098         #1-/ .code={%
2099             \pgfmathtruncatemacro\forest@temp{##1}%
2100             \c@pgf@counta=\forestrve{#3}\relax
2101             \advance\c@pgf@counta-\forest@temp\relax
2102             \forestreset{#3}{\the\c@pgf@counta}%
2103         },
2104         #1*/.code={%
2105             \pgfmathtruncatemacro\forest@temp{##1}%
2106             \c@pgf@counta=\forestrve{#3}\relax

```



```

2107     \multiply\c@pgf@counta\forest@temp\relax
2108     \forestreset{#3}{\the\c@pgf@counta}%
2109 },
2110 #1/.code={%
2111     \pgfmathtruncatemacro\forest@temp{##1}%
2112     \c@pgf@counta=\forestrve{#3}\relax
2113     \divide\c@pgf@counta\forest@temp\relax
2114     \forestreset{#3}{\the\c@pgf@counta}%
2115 },
2116 #1'/.code={%
2117     \c@pgf@counta=##1\relax
2118     \forestreset{#3}{\the\c@pgf@counta}%
2119 },
2120 #1'+/.code={%
2121     \c@pgf@counta=\forestrve{#3}\relax
2122     \advance\c@pgf@counta##1\relax
2123     \forestreset{#3}{\the\c@pgf@counta}%
2124 },
2125 #1'-/.code={%
2126     \c@pgf@counta=\forestrve{#3}\relax
2127     \advance\c@pgf@counta-##1\relax
2128     \forestreset{#3}{\the\c@pgf@counta}%
2129 },
2130 #1'*/.style={%
2131     \c@pgf@counta=\forestrve{#3}\relax
2132     \multiply\c@pgf@counta##1\relax
2133     \forestreset{#3}{\the\c@pgf@counta}%
2134 },
2135 #1':/.style={%
2136     \c@pgf@counta=\forestrve{#3}\relax
2137     \divide\c@pgf@counta##1\relax
2138     \forestreset{#3}{\the\c@pgf@counta}%
2139 },
2140 }%
2141 \pgfkeyssetvalue{#1/option@name}{#3}%
2142 \pgfkeyssetvalue{#1+/option@name}{#3}%
2143 \pgfkeyssetvalue{#1-/option@name}{#3}%
2144 \pgfkeyssetvalue{#1*/option@name}{#3}%
2145 \pgfkeyssetvalue{#1:/option@name}{#3}%
2146 \pgfkeyssetvalue{#1'/option@name}{#3}%
2147 \pgfkeyssetvalue{#1'+/option@name}{#3}%
2148 \pgfkeyssetvalue{#1'-/option@name}{#3}%
2149 \pgfkeyssetvalue{#1'* /option@name}{#3}%
2150 \pgfkeyssetvalue{#1':/option@name}{#3}%
2151 }
2152 \def\forest@declarebooleanregister@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmathname
2153     \pgfkeysalso{%
2154         #1/.code={%
2155             \ifstrequal{##1}{1}{%
2156                 \forestreset{#3}{1}%
2157             }{%
2158                 \ifstrequal{##1}{0}{%
2159                     \forestreset{#3}{0}%
2160                 }{%
2161                     \pgfmathifthenelse{##1}{1}{0}%
2162                     \forestrlet{#3}\pgfmathresult
2163                 }%
2164             }%
2165         },
2166         #1/.default=1,
2167         #2/not #3/.code={\forestreset{#3}{0}},

```

```

2168     #2/if #3/.code 2 args={%
2169         \foresttrget{#3}\forest@temp@option@value
2170         \ifnum\forest@temp@option@value=1
2171             \pgfkeysalso{##1}%
2172         \else
2173             \pgfkeysalso{##2}%
2174         \fi
2175     },
2176 }%
2177 \pgfkeyssetvalue{#1/option@name}{#3}%
2178 \pgfmathdeclarefunction{#4}{1}{\forest@pgfmathhelper@register@count{##1}{#3}}%
2179 }
2180 \forestset{
2181   declare toks register/.code={%
2182     \forest@declareregisterhandler\forest@declaretokregister@handler{#1}%
2183   },
2184   declare autowrapped toks register/.code={%
2185     \forest@declareregisterhandler\forest@declareautowrappedtokregister@handler{#1}%
2186   },
2187   declare keylist register/.code={%
2188     \forest@declareregisterhandler\forest@declarekeylistregister@handler{#1}%
2189   },
2190   declare dimen register/.code={%
2191     \forest@declareregisterhandler\forest@declaredimenregister@handler{#1}%
2192   },
2193   declare count register/.code={%
2194     \forest@declareregisterhandler\forest@declarecountregister@handler{#1}%
2195   },
2196   declare boolean register/.code={%
2197     \forest@declareregisterhandler\forest@declarebooleanregister@handler{#1}%
2198   },
2199 }

```

Declare some temporary registers.

```

2200 \forestset{
2201   declare toks register=temptoksa,temptoksa={},
2202   declare toks register=temptoksb,temptoksb={},
2203   declare toks register=temptoksc,temptoksc={},
2204   declare toks register=temptoksd,temptoksd={},
2205   declare keylist register=tempkeylista,tempkeylista'={},
2206   declare keylist register=tempkeylistb,tempkeylistb'={},
2207   declare keylist register=tempkeylistc,tempkeylistc'={},
2208   declare keylist register=tempkeylistd,tempkeylistd'={},
2209   declare dimen register=tempdima,tempdima'={0pt},
2210   declare dimen register=tempdimb,tempdimb'={0pt},
2211   declare dimen register=tempdimc,tempdimc'={0pt},
2212   declare dimen register=tempdimd,tempdimd'={0pt},
2213   declare dimen register=tempdimx,tempdimx'={0pt},
2214   declare dimen register=tempdimy,tempdimy'={0pt},
2215   declare dimen register=tempdiml,tempdiml'={0pt},
2216   declare dimen register=tempdims,tempdims'={0pt},
2217   declare count register=tempcounta,tempcounta'={0},
2218   declare count register=tempcountb,tempcountb'={0},
2219   declare count register=tempcountc,tempcountc'={0},
2220   declare count register=tempcountd,tempcountd'={0},
2221   declare boolean register=tempboola,tempboola={0},
2222   declare boolean register=tempboolb,tempboolb={0},
2223   declare boolean register=tempboolc,tempboolc={0},
2224   declare boolean register=tempboold,tempboold={0},
2225 }

```

6.3.3 Declaring options

```

2226 \def\forest@node@Nametoid#1{% #1 = name
2227   \csname forest@id@of@#1\endcsname
2228 }
2229 \def\forest@node@ifnamedefined#1#2#3{% #1 = name, #2=true,#3=false
2230   \ifcsvoid{forest@id@of@#1}{#3}{#2}%
2231 }
2232 \def\forest@node@setname#1{%
2233   \def\forest@temp@setname{y}%
2234   \def\forest@temp@silent{n}%
2235   \def\forest@temp@propagating{n}%
2236   \forest@node@setnameoralias{#1}%
2237 }
2238 \def\forest@node@setname@silent#1{%
2239   \def\forest@temp@setname{y}%
2240   \def\forest@temp@silent{y}%
2241   \def\forest@temp@propagating{n}%
2242   \forest@node@setnameoralias{#1}%
2243 }
2244 \def\forest@node@setalias#1{%
2245   \def\forest@temp@setname{n}%
2246   \def\forest@temp@silent{n}%
2247   \def\forest@temp@propagating{n}%
2248   \forest@node@setnameoralias{#1}%
2249 }
2250 \def\forest@node@setalias@silent#1{%
2251   \def\forest@temp@setname{n}%
2252   \def\forest@temp@silent{y}%
2253   \def\forest@temp@propagating{n}%
2254   \forest@node@setnameoralias{#1}%
2255 }
2256 \def\forest@node@setnameoralias#1{%
2257   \ifstrempy{#1}{%
2258     \forest@node@setnameoralias{node@\forest@cn}%
2259   }{%
2260     \forest@node@ifnamedefined{#1}{%
2261       \if y\forest@temp@propagating
2262         % this will find a unique name, eventually:
2263         \@escapeif{\forest@node@setnameoralias{#1@\forest@cn}}%
2264       \else\@escapeif{%
2265         \if y\forest@temp@setname
2266           \@escapeif{\forest@node@setnameoralias@nameclash{#1}}%
2267         \else\@escapeif{% setting an alias: clashing with alias is not a problem
2268           \forest@get{\forest@node@Nametoid{#1}}{name}\forest@temp
2269           \expandafter\ifstrequal\expandafter{\forest@temp}{#1}{%
2270             \forest@node@setnameoralias@nameclash{#1}%
2271           }{%
2272             \forest@node@setnameoralias@do{#1}%
2273           }%
2274         }\fi
2275       }\fi
2276     }{%
2277       \forest@node@setnameoralias@do{#1}%
2278     }%
2279   }%
2280 }
2281 \def\forest@node@setnameoralias@nameclash#1{%
2282   \if y\forest@temp@silent
2283     \forest@fornode{\forest@node@Nametoid{#1}}{%
2284       \def\forest@temp@propagating{y}%

```

```

2285     \forest@node@setnameoralias{}%
2286 }%
2287 \forest@node@setnameoralias@do{#1}%
2288 \else
2289   \PackageError{forest}{Node name "#1" is already used}{}%
2290 \fi
2291 }
2292 \def\forest@node@setnameoralias@do#1{%
2293   \if y\forest@temp@setname
2294     \csdef{forest@id@of@\forest@ve{name}}{}%
2295     \forest@set{name}{#1}%
2296   \fi
2297   \csdef{forest@id@of@#1}{\forest@cn}%
2298 }
2299 \forestset{
2300   TeX/.code={#1},
2301   TeX'/.code={\appto\forest@externalize@loadimages{#1}{#1},
2302   TeX''/.code={\appto\forest@externalize@loadimages{#1}},
2303   options/.code={\forestset{#1}},
2304   typeout/.style={TeX={\typeout{#1}}},
2305   declare toks={name}{},
2306   name/.code={% override the default setter
2307     \forest@for@node{\forest@setter@node}{\forest@node@setname{#1}}%
2308   },
2309   name/.default={},
2310   name'/.code={% override the default setter
2311     \forest@for@node{\forest@setter@node}{\forest@node@setname@silent{#1}}%
2312   },
2313   name'/.default={},
2314   alias/.code={\forest@for@node{\forest@setter@node}{\forest@node@setalias{#1}}},
2315   alias'/.code={\forest@for@node{\forest@setter@node}{\forest@node@setalias@silent{#1}}},
2316   begin draw/.code={\begin{tikzpicture}},
2317   end draw/.code={\end{tikzpicture}},
2318   declare keylist register=default preamble,
2319   default preamble'={},
2320   declare keylist register=preamble,
2321   preamble'={},
2322   declare autowrapped toks={content}{},
2323   % #1 = which option to split, #2 = separator (one char!), #3 = receiving options
2324   %%% begin listing region: split_option
2325   split option/.style n args=3{split/.process args={0}{#1}{#2}{#3}}
2326   %%% end listing region: split_option
2327   ,
2328   split register/.style n args=3{% #1 = which register to split, #2 = separator (one char!), #3 = receiving options
2329     split/.process args={R}{#1}{#2}{#3},
2330   },
2331   TeX={%
2332     \def\forest@split@sourcevalues{}%
2333     \def\forest@split@sourcevalue{}%
2334     \def\forest@split@receivingoptions{}%
2335     \def\forest@split@receivingoption{}%
2336   },
2337   split/.code n args=3{% #1 = string to split, #2 = separator (one char!), #3 = receiving options
2338     \forest@saveandrestoremacro\forest@split@sourcevalues{%
2339       \forest@saveandrestoremacro\forest@split@sourcevalue{%
2340         \forest@saveandrestoremacro\forest@split@receivingoptions{%
2341           \forest@saveandrestoremacro\forest@split@receivingoption{%
2342             \def\forest@split@sourcevalues{#1#2}%
2343             \edef\forest@split@receivingoptions{#3}%
2344             \def\forest@split@receivingoption{}%
2345           \safeloop

```

```

2346 \expandafter\forest@split\expandafter{\forest@split@sourcevalues}{#2}\forest@split@sourcevalue\
2347 \ifdefemptyforest@split@receivingoptions{}\{%
2348 \expandafter\forest@split\expandafter{\forest@split@receivingoptions}{,}\forest@temp\forest@s
2349 \ifdefemptyforest@temp{}\let\forest@split@receivingoption\forest@temp\def\forest@temp{}\}%
2350 }%
2351 \edef\forest@marshal{%
2352 \noexpand\pgfkeysalso{\forest@split@receivingoption={\expandonce{\forest@split@sourcevalue}}}%
2353 }\forest@marshal
2354 \ifdefemptyforest@split@sourcevalues{\forest@tempfalse}\forest@temptrue}%
2355 \ifforest@temp
2356 \saferepeat
2357 }}}}%,
2358 },
2359 declare count={grow}{270},
2360 TeX={% a hack for grow-reversed connection, and compass-based grow specification
2361 \forest@copycommandkey{/forest/grow}{/forest/grow@}%
2362 %\pgfkeysgetvalue{/forest/grow/.@cmd}\forest@temp
2363 %\pgfkeyslet{/forest/grow@/.@cmd}\forest@temp
2364 },
2365 grow/.style={grow@={#1},reversed=0},
2366 grow'/.style={grow@={#1},reversed=1},
2367 grow''/.style={grow@={#1}},
2368 grow@/.is choice,
2369 grow@/east/.style={/forest/grow@=0},
2370 grow@/north east/.style={/forest/grow@=45},
2371 grow@/north/.style={/forest/grow@=90},
2372 grow@/north west/.style={/forest/grow@=135},
2373 grow@/west/.style={/forest/grow@=180},
2374 grow@/south west/.style={/forest/grow@=225},
2375 grow@/south/.style={/forest/grow@=270},
2376 grow@/south east/.style={/forest/grow@=315},
2377 grow@/.unknown/.code={\let\forest@temp@grow\pgfkeyscurrentname
2378 \pgfkeysalso{/forest/grow@/.expand once=\forest@temp@grow}},
2379 declare boolean={reversed}{0},
2380 declare toks={parent anchor}{},
2381 declare toks={child anchor}{},
2382 declare toks={anchor}{base},
2383 Autoforward={anchor}{
2384 node options-=anchor,
2385 node options+=\{anchor={#1}\}
2386 },
2387 anchor'/.style={anchor@no@compass=true,anchor=#1},
2388 anchor+/.style={anchor@no@compass=true,anchor+=#1},
2389 anchor-/.style={anchor@no@compass=true,anchor-=#1},
2390 anchor*/.style={anchor@no@compass=true,anchor*=#1},
2391 anchor:/.style={anchor@no@compass=true,anchor:=#1},
2392 anchor'+/.style={anchor@no@compass=true,anchor'+=#1},
2393 anchor'-/.style={anchor@no@compass=true,anchor'-=#1},
2394 anchor'*/.style={anchor@no@compass=true,anchor'*=#1},
2395 anchor':/.style={anchor@no@compass=true,anchor':=#1},
2396 % /tikz/forest anchor/.style={
2397 % /forest/TeX={\forestanchortotikzanchor{#1}\forest@temp@anchor},
2398 % anchor/.expand once=\forest@temp@anchor
2399 % },
2400 declare toks={calign}{midpoint},
2401 TeX={%
2402 \forest@copycommandkey{/forest/calign}{/forest/calign'}%
2403 },
2404 calign/.is choice,
2405 calign/child/.style={calign'=child},
2406 calign/first/.style={calign'=child,calign primary child=1},

```

```

2407 calign/last/.style={calign'=child,calign primary child=-1},
2408 calign with current/.style={for parent/.wrap pgfmath arg={calign=child,calign primary child=##1}{n}},
2409 calign with current edge/.style={for parent/.wrap pgfmath arg={calign=child edge,calign primary child=##1}{n}},
2410 calign/child edge/.style={calign'=child edge},
2411 calign/midpoint/.style={calign'=midpoint},
2412 calign/center/.style={calign'=midpoint,calign primary child=1,calign secondary child=-1},
2413 calign/edge midpoint/.style={calign'=edge midpoint},
2414 calign/fixed angles/.style={calign'=fixed angles},
2415 calign/fixed edge angles/.style={calign'=fixed edge angles},
2416 calign/.unknown/.code={\PackageError{forest}{unknown calign '\pgfkeyscurrentname'}{}}},
2417 declare count={calign primary child}{1},
2418 declare count={calign secondary child}{-1},
2419 declare count={calign primary angle}{-35},
2420 declare count={calign secondary angle}{35},
2421 calign child/.style={calign primary child={#1}},
2422 calign angle/.style={calign primary angle={-#1},calign secondary angle={#1}},
2423 declare toks={tier}{},
2424 declare toks={fit}{tight},
2425 declare boolean={ignore}{0},
2426 declare boolean={ignore edge}{0},
2427 no edge/.style={edge'={},ignore edge},
2428 declare keylist={edge}{draw},
2429 declare toks={edge path}{%
2430   \noexpand\path[\forestoption{edge}]{%
2431     (\forestOve{\forestove{@parent}}{name}.parent anchor)--(\forestove{name}.child anchor)
2432     % =
2433     % (!u.parent anchor)--(.child anchor)\forestoption{edge label};
2434     \forestoption{edge label}};
2435 },
2436 edge path'/.style={
2437   edge path={%
2438     \noexpand\path[\forestoption{edge}]{%
2439       #1%
2440       \forestoption{edge label};
2441     }
2442 },
2443 declare toks={edge label}{},
2444 declare boolean={phantom}{0},
2445 baseline/.style={alias={forest@baseline@node}},
2446 declare readonly count={n},
2447 declare readonly count={n'},
2448 declare readonly count={n children},
2449 declare readonly count={level},
2450 declare dimen=x{0pt},
2451 declare dimen=y{0pt},
2452 declare dimen={s}{0pt},
2453 declare dimen={l}{6ex}, % just in case: should be set by the calibration
2454 declare dimen={s sep}{0.6666em},
2455 declare dimen={l sep}{1ex}, % just in case: calibration!
2456 declare keylist={node options}{anchor=base},
2457 declare toks={tikz}{},
2458 afterthought/.style={tikz+=#{#1}},
2459 label/.style={tikz={\path[late options={%
2460   name=\forestoption{name},label={#1}}];}},
2461 pin/.style={tikz={\path[late options={%
2462   name=\forestoption{name},pin={#1}}];}},
2463 declare toks={content format}{\forestoption{content}},
2464 plain content/.style={content format={\forestoption{content}}},
2465 math content/.style={content format={\noexpand\ensuremath{\forestoption{content}}}},
2466 declare toks={node format}{%
2467   \noexpand\node

```

```

2468 (\forestoption{name})%
2469 [\forestoption{node options}]%
2470 {\forestoption{content format}};%
2471 },
2472 node format'/.style={
2473   node format={\noexpand\node(\forestoption{name})#1;}
2474 },
2475 tabular@environment/.style={content format={%
2476   \noexpand\begin{tabular}[\forestoption{base}]{\forestoption{align}}%
2477   \forestoption{content}%
2478   \noexpand\end{tabular}%
2479 }},
2480 declare toks={align}{},
2481 TeX={%
2482   \forest@copycommandkey{/forest/align}{/forest/align'}%
2483   %\pgfkeysgetvalue{/forest/align/.@cmd}\forest@temp
2484   %\pgfkeyslet{/forest/align'/.@cmd}\forest@temp
2485 },
2486 align/.is choice,
2487 align/.unknown/.code={%
2488   \edef\forest@marshal{%
2489     \noexpand\pgfkeysalso{%
2490       align'={\pgfkeyscurrentname},%
2491       tabular@environment
2492     }%
2493   }\forest@marshal
2494 },
2495 align/center/.style={align'={@{c@{}}},tabular@environment},
2496 align/left/.style={align'={@{l@{}}},tabular@environment},
2497 align/right/.style={align'={@{r@{}}},tabular@environment},
2498 declare toks={base}{t},
2499 TeX={%
2500   \forest@copycommandkey{/forest/base}{/forest/base'}%
2501   %\pgfkeysgetvalue{/forest/base/.@cmd}\forest@temp
2502   %\pgfkeyslet{/forest/base'/.@cmd}\forest@temp
2503 },
2504 base/.is choice,
2505 base/top/.style={base'=t},
2506 base/bottom/.style={base'=b},
2507 base/.unknown/.style={base'/.expand once=\pgfkeyscurrentname},
2508 unknown to/.store in=\forest@unknownto,
2509 unknown to=node options,
2510 unknown key error/.code={\PackageError{forest}{Unknown keyval: \detokenize{#1}}{}},
2511 content to/.store in=\forest@contentto,
2512 content to=content,
2513 .unknown/.code={%
2514   \expandafter\pgfutil@in@\expandafter.\expandafter{\pgfkeyscurrentname}%
2515   \ifpgfutil@in@
2516     \expandafter\forest@relatednode@option@setter\pgfkeyscurrentname=#1\forest@END
2517   \else
2518     \edef\forest@marshal{%
2519       \noexpand\pgfkeysalso{\forest@unknownto={\pgfkeyscurrentname=\unexpanded{#1}}}%
2520     }\forest@marshal
2521   \fi
2522 },
2523 get node boundary/.code={%
2524   \foresttoget{@boundary}\forest@node@boundary
2525   \def#1{}%
2526   \forest@extendpath#1\forest@node@boundary{\pgfpoint{\forestove{x}}{\forestove{y}}}%
2527 },
2528 % get min 1 tree boundary/.code={%

```

```

2529 % \forest@get@tree@boundary{negative}{\the\numexpr\forestove{grow}-90\relax}#1},
2530 % get max l tree boundary/.code={%
2531 % \forest@get@tree@boundary{positive}{\the\numexpr\forestove{grow}-90\relax}#1},
2532 get min s tree boundary/.code={%
2533 \forest@get@tree@boundary{negative}{\forestove{grow}}#1},
2534 get max s tree boundary/.code={%
2535 \forest@get@tree@boundary{positive}{\forestove{grow}}#1},
2536 use as bounding box/.style={%
2537 before drawing tree={
2538 tikz+/.expanded={%
2539 \noexpand\pgfresetboundingbox
2540 \noexpand\useasboundingbox
2541 ($(.anchor)+(\forestoption{min x},\forestoption{min y}))$)
2542 rectangle
2543 ($(.anchor)+(\forestoption{max x},\forestoption{max y}))$)
2544 ;
2545 }
2546 }
2547 },
2548 use as bounding box'/.style={%
2549 before drawing tree={
2550 tikz+/.expanded={%
2551 \noexpand\pgfresetboundingbox
2552 \noexpand\useasboundingbox
2553 ($(.anchor)+(\forestoption{min x}+\pgfkeysvalueof{/pgf/outer xsep}/2+\pgfkeysvalueof{/pgf/inner xsep}))$)
2554 rectangle
2555 ($(.anchor)+(\forestoption{max x}-\pgfkeysvalueof{/pgf/outer xsep}/2-\pgfkeysvalueof{/pgf/inner xsep}))$)
2556 ;
2557 }
2558 }
2559 },
2560 }%
2561 \def\forest@iftikzkey#1#2#3{% #1 = key name, #2 = true code, #3 = false code
2562 \forest@temptrue
2563 \pgfkeysifdefined{/tikz/\pgfkeyscurrentname}{\}%
2564 \pgfkeysifdefined{/tikz/\pgfkeyscurrentname/.@cmd}{\}%
2565 \pgfkeysifdefined{/pgf/\pgfkeyscurrentname}{\}%
2566 \pgfkeysifdefined{/pgf/\pgfkeyscurrentname/.@cmd}{\}%
2567 \forest@tempfalse
2568 }}}}
2569 \ifforest@temp\@escapeif{#2}\else\@escapeif{#3}\fi
2570 }
2571 \def\forest@ifoptionortikzkey#1#2#3{% #1 = key name, #2 = true code, #3 = false code
2572 \forest@temptrue
2573 \pgfkeysifdefined{/forest/\pgfkeyscurrentname}{\}%
2574 \pgfkeysifdefined{/forest/\pgfkeyscurrentname/.@cmd}{\}%
2575 \forest@iftikzkey{#1}{\}\}%
2576 }}}}
2577 \ifforest@temp\@escapeif{#2}\else\@escapeif{#3}\fi
2578 }
2579 \def\forest@get@tree@boundary#1#2#3{% #1=pos/neg, #2=grow, #3=receiving cs
2580 \def#3{}%
2581 \forest@node@getedge{#1}{#2}\forest@temp@boundary
2582 \forest@extendpath#3\forest@temp@boundary{\pgfpoint{\forestove{x}}{\forestove{y}}}%
2583 }
2584 \def\forest@setter@node{\forest@cn}%
2585 \def\forest@relatednode@option@compat@ignoreinvalidsteps#1{#1}
2586 \def\forest@relatednode@option@setter#1.#2=#3\forest@END{%
2587 \forest@forthis{%
2588 \forest@relatednode@option@compat@ignoreinvalidsteps{%
2589 \forest@nameandgo{#1}%

```



```

2590     \let\forest@setter@node\forest@cn
2591   }%
2592 }%
2593 \ifnum\forest@setter@node=0
2594   \else
2595     \forestset{#2={#3}}%
2596   \fi
2597 \def\forest@setter@node{\forest@cn}%
2598 }%
2599 \def\forest@split#1#2#3#4{% #1=list (assuming that the list is nonempty and finishes with the separator), #2
2600   \def\forest@split@@##1#2##2\forest@split@@##3##4{\def##3{##1}\def##4{##2}}%
2601   \forest@split@@#1\forest@split@@{#3}{#4}}

```

6.3.4 Option propagation

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

```

2602 \forestset{
2603   for tree'/.style 2 args={#1,for children={for tree'={#1}{#2}},#2},
2604   if/.code n args={3}{%
2605     \pgfmathparse{#1}%
2606     \ifnum\pgfmathresult=0
2607       \escapeif{\pgfkeysalso{#3}}%
2608     \else
2609       \escapeif{\pgfkeysalso{#2}}%
2610     \fi
2611   },
2612   if nodewalk valid/.code n args={3}{%
2613     \edef\forest@marshal{%
2614       \unexpanded{\forest@forthis{%
2615         \forest@nodewalk{%
2616           on invalid={fake}{#1},
2617           TeX={\global\let\forest@global@temp\forest@cn}
2618         }}%
2619       }%
2620     \def\forest@cn{\forest@cn}\unexpanded{%
2621     \ifnum\forest@global@temp=0
2622       \escapeif{\pgfkeysalso{#3}}%
2623     \else
2624       \escapeif{\pgfkeysalso{#2}}%
2625     \fi}%
2626     }\forest@marshal
2627   },
2628   if nodewalk empty/.code n args={3}{%
2629     \forest@forthis{%
2630       \forest@nodewalk{%
2631         on invalid={fake}{#1},
2632         TeX={\global\let\forest@global@temp\forest@nodewalk@cn},
2633       }}%
2634     }%
2635     \ifnum\forest@global@temp=0
2636       \escapeif{\pgfkeysalso{#2}}%
2637     \else
2638       \escapeif{\pgfkeysalso{#3}}%
2639     \fi
2640   },
2641   where/.style n args={3}{for tree={if={#1}{#2}{#3}}},
2642   where nodewalk valid/.style n args={3}{for tree={if nodewalk valid={#1}{#2}{#3}}},
2643   where nodewalk empty/.style n args={3}{for tree={if nodewalk empty={#1}{#2}{#3}}},
2644   repeat/.code 2 args={%
2645     \pgfmathtruncatemacro\forest@temp{#1}%
2646     \expandafter\forest@repeatkey\expandafter{\forest@temp}{#2}%

```

```

2647 },
2648 until/.code 2 args={%
2649   \ifstrempy{#1}{%
2650     \forest@untilkey{\ifnum\forest@cn=0\relax\forestloopbreak\fi}{on invalid={fake}{#2}}%
2651   }{%
2652     \forest@untilkey{\pgfmathifthenelse{#1}{"\noexpand\forestloopbreak"}{""}\pgfmathresult}{#2}%
2653   }%
2654 },
2655 while/.code 2 args={%
2656   \ifstrempy{#1}{%
2657     \forest@untilkey{\ifnum\forest@cn=0\relax\forestloopbreak\fi}{on invalid={fake}{#2}}%
2658   }{%
2659     \forest@untilkey{\pgfmathifthenelse{not(#1)}{"\noexpand\forestloopbreak"}{""}\pgfmathresult}{#2}%
2660   }%
2661 },
2662 do until/.code 2 args={%
2663   \ifstrempy{#1}{%
2664     \forest@dountilkey{\ifnum\forest@cn=0\relax\forestloopbreak\fi}{on invalid={fake}{#2}}%
2665   }{%
2666     \forest@dountilkey{\pgfmathifthenelse{#1}{"\noexpand\forestloopbreak"}{""}\pgfmathresult}{#2}%
2667   }%
2668 },
2669 do while/.code 2 args={%
2670   \ifstrempy{#1}{%
2671     \forest@dountilkey{\ifnum\forest@cn=0\relax\forestloopbreak\fi}{on invalid={fake}{#2}}%
2672   }{%
2673     \forest@dountilkey{\pgfmathifthenelse{not(#1)}{"\noexpand\forestloopbreak"}{""}\pgfmathresult}{#2}%
2674   }%
2675 },
2676 until nodewalk valid/.code 2 args={%
2677   \forest@untilkey{\forest@forthis{%
2678     \forest@nodewalk{on invalid={fake}{#1},TeX={\ifnum\forest@cn=0\relax\else\forestloopbreak\fi}}{}}{#2}%
2679 },
2680 while nodewalk valid/.code 2 args={%
2681   \forest@untilkey{\forest@forthis{%
2682     \forest@nodewalk{on invalid={fake}{#1},TeX={\ifnum\forest@cn=0\relax\forestloopbreak\fi}}{}}{#2}%
2683 },
2684 do until nodewalk valid/.code 2 args={%
2685   \forest@dountilkey{\forest@forthis{%
2686     \forest@nodewalk{on invalid={fake}{#1},TeX={\ifnum\forest@cn=0\relax\else\forestloopbreak\fi}}{}}{#2}%
2687 },
2688 do while nodewalk valid/.code 2 args={%
2689   \forest@dountilkey{\forest@forthis{%
2690     \forest@nodewalk{on invalid={fake}{#1},TeX={\ifnum\forest@cn=0\relax\forestloopbreak\fi}}{}}{#2}%
2691 },
2692 until nodewalk empty/.code 2 args={%
2693   \forest@untilkey{\forest@forthis{%
2694     \forest@nodewalk{on invalid={fake}{#1},TeX={\ifnum\forest@nodewalk@n=0\relax\forestloopbreak\fi}}{}}{#2}%
2695 },
2696 while nodewalk empty/.code 2 args={%
2697   \forest@untilkey{\forest@forthis{%
2698     \forest@nodewalk{on invalid={fake}{#1},TeX={\ifnum\forest@nodewalk@n=0\relax\else\forestloopbreak\fi}}{}}{#2}%
2699 },
2700 do until nodewalk empty/.code 2 args={%
2701   \forest@dountilkey{\forest@forthis{%
2702     \forest@nodewalk{on invalid={fake}{#1},TeX={\ifnum\forest@nodewalk@n=0\relax\forestloopbreak\fi}}{}}{#2}%
2703 },
2704 do while nodewalk empty/.code 2 args={%
2705   \forest@dountilkey{\forest@forthis{%
2706     \forest@nodewalk{on invalid={fake}{#1},TeX={\ifnum\forest@nodewalk@n=0\relax\else\forestloopbreak\fi}}{}}{#2}%
2707 },

```

```

2708 break/.code={\forestloopBreak{#1}},
2709 break/.default=0,
2710 }
2711 \def\forest@repeatkey#1#2{%
2712   \safeRKloop
2713   \ifnum\safeRKloopn>#1\relax
2714   \csuse{safeRKbreak@\the\safeRKloop@depth true}%
2715   \fi
2716   \expandafter\unless\csname ifsafeRKbreak@\the\safeRKloop@depth\endcsname
2717   \pgfkeysalso{#2}%
2718   \safeRKrepeat
2719 }
2720 \def\forest@untilkey#1#2{% #1 = condition, #2 = keys
2721   \safeRKloop
2722   #1%
2723   \expandafter\unless\csname ifsafeRKbreak@\the\safeRKloop@depth\endcsname
2724   \pgfkeysalso{#2}%
2725   \safeRKrepeat
2726 }
2727 \def\forest@dountilkey#1#2{% #1 = condition, #2 = keys
2728   \safeRKloop
2729   \pgfkeysalso{#2}%
2730   #1%
2731   \expandafter\unless\csname ifsafeRKbreak@\the\safeRKloop@depth\endcsname
2732   \safeRKrepeat
2733 }
2734 \def\forestloopbreak{%
2735   \csname safeRKbreak@\the\safeRKloop@depth true\endcsname
2736 }
2737 \def\forestloopBreak#1{%
2738   \csname safeRKbreak@\number\numexpr\the\safeRKloop@depth-#1\relax true\endcsname
2739 }
2740 \def\forestloopcount{%
2741   \csname safeRKloopn@\number\numexpr\the\safeRKloop@depth\endcsname
2742 }
2743 \def\forestloopCount#1{%
2744   \csname safeRKloopn@\number\numexpr\the\safeRKloop@depth-#1\endcsname
2745 }
2746 \pgfmathdeclarefunction{forestloopcount}{1}{%
2747   \edef\pgfmathresult{\forestloopCount{\ifstreamp{#1}{0}{#1}}}%
2748 }
2749 \forest@copycommandkey{/forest/repeat}{/forest/nodewalk/repeat}
2750 \forest@copycommandkey{/forest/while}{/forest/nodewalk/while}
2751 \forest@copycommandkey{/forest/do while}{/forest/nodewalk/do while}
2752 \forest@copycommandkey{/forest/until}{/forest/nodewalk/until}
2753 \forest@copycommandkey{/forest/do until}{/forest/nodewalk/do until}
2754 \forest@copycommandkey{/forest/if}{/forest/nodewalk/if}
2755 \forest@copycommandkey{/forest/if nodewalk valid}{/forest/nodewalk/if nodewalk valid}
2756 %

```

6.4 Aggregate functions

```

2757 \forestset{
2758   aggregate postparse/.is choice,
2759   aggregate postparse/int/.code={%
2760     \let\forest@aggregate@pgfmathpostparse\forest@aggregate@pgfmathpostparse@toint},
2761   aggregate postparse/none/.code={%
2762     \let\forest@aggregate@pgfmathpostparse\relax},
2763   aggregate postparse/print/.code={%
2764     \let\forest@aggregate@pgfmathpostparse\forest@aggregate@pgfmathpostparse@print},

```

```

2765 aggregate postparse/macro/.code={%
2766   \let\forest@aggregate@pgfmthpostparse\forest@aggregate@pgfmthpostparse@usemacro},
2767 aggregate postparse macro/.store in=\forest@aggregate@pgfmthpostparse@macro,
2768 }
2769 \def\forest@aggregate@pgfmthpostparse@print{%
2770   \pgfmthprintnumbertto{\pgfmthresult}{\pgfmthresult}%
2771 }
2772 \def\forest@aggregate@pgfmthpostparse@toint{%
2773   \expandafter\forest@split\expandafter{\pgfmthresult.}{.}\pgfmthresult\forest@temp
2774 }
2775 \def\forest@aggregate@pgfmthpostparse@usemacro{%
2776   \forest@aggregate@pgfmthpostparse@macro
2777 }
2778 \let\forest@aggregate@pgfmthpostparse\relax
2779 \pgfkeys{
2780   /handlers/.aggregate/.code n args=4{%
2781     % #1 = start value
2782     % #2 = pgfmth expression for every step
2783     % #3 = pgfmth expression for after walk
2784     % #4 = nodewalk
2785     \edef\forest@marshal{%
2786       \unexpanded{\forest@aggregate{#1}{#2}{#3}{#4}}{\pgfkeyscurrentpath}%
2787     }\forest@marshal
2788   },
2789   /handlers/.sum/.style 2 args={/handlers/.aggregate={0}{(##1)+(##1)}{##1}{#2}},
2790   /handlers/.count/.style={/handlers/.aggregate={0}{1+(##1)}{##1}{#1}},
2791   /handlers/.average/.style 2 args={/handlers/.aggregate={0}{(##1)+(##1)}{##1/\forestregister{aggregate n}}{#2}},
2792   /handlers/.product/.style 2 args={/handlers/.aggregate={1}{(##1)*(##1)}{##1}{#2}},
2793   /handlers/.min/.style 2 args={/handlers/.aggregate={}{min(##1,##1)}{##1}{#2}},
2794   /handlers/.max/.style 2 args={/handlers/.aggregate={}{max(##1,##1)}{##1}{#2}},
2795   /forest/declare count register={aggregate n},
2796 }
2797
2798 \def\forest@aggregate#1#2#3#4#5{%
2799   % #5 = current path
2800   \def\forest@aggregate@result{#1}%
2801   \forest@forthis{%
2802     \foreststrset{aggregate n}{0}%
2803     \forest@nodewalk{#4}{%
2804       TeX={%
2805         \forestreset{aggregate n}{\number\numexpr\forestrv{aggregate n}+1}%
2806         \def\forest@marshal##1{\pgfmthparse{#2}}%
2807         \expandafter\forest@marshal\expandafter{\forest@aggregate@result}%
2808         \let\forest@aggregate@result\pgfmthresult
2809       }%
2810     }{}%
2811   }%
2812   \def\forest@marshal##1{\pgfmthparse{#3}}%
2813   \expandafter\forest@marshal\expandafter{\forest@aggregate@result}%
2814   \let\forest@aggregate@result\pgfmthresult
2815   \let\forest@temp@pgfmthpostparse\pgfmthpostparse
2816   \let\pgfmthpostparse\forest@aggregate@pgfmthpostparse
2817   \pgfmthqparse{\forest@aggregate@result pt}%
2818   \let\pgfmthpostparse\forest@temp@pgfmthpostparse
2819   \let\forest@aggregate@result\pgfmthresult
2820   \pgfkeysalso{#5/.expand once=\forest@aggregate@result}%
2821 }

```

6.4.1 pgfmth extensions

```

2822 \pgfmthdeclarefunction{strequal}{2}{%
2823   \ifstrequal{#1}{#2}{\def\pgfmthresult{1}}{\def\pgfmthresult{0}}%

```

```

2824 }
2825 \pgfmathdeclarefunction{instr}{2}{%
2826   \pgfutil@in@{#1}{#2}%
2827   \ifpgfutil@in@{\def\pgfmathresult{1}\else\def\pgfmathresult{0}\fi
2828 }
2829 \pgfmathdeclarefunction{strcat}{...}{%
2830   \edef\pgfmathresult{\forest@strip@braces{#1}}%
2831 }
2832 \pgfmathdeclarefunction{min_s}{2}{% #1 = node, #2 = context node (for growth rotation)
2833   \forest@forthis{%
2834     \forest@nameandgo{#1}%
2835     \forest@compute@minmax@ls{#2}%
2836     \pgfmathsetmacro\pgfmathresult{\forestove{min@s}}%
2837   }%
2838 }
2839 \pgfmathdeclarefunction{min_l}{2}{% #1 = node, #2 = context node (for growth rotation)
2840   \forest@forthis{%
2841     \forest@nameandgo{#1}%
2842     \forest@compute@minmax@ls{#2}%
2843     \pgfmathsetmacro\pgfmathresult{\forestove{min@l}}%
2844   }%
2845 }
2846 \pgfmathdeclarefunction{max_s}{2}{% #1 = node, #2 = context node (for growth rotation)
2847   \forest@forthis{%
2848     \forest@nameandgo{#1}%
2849     \forest@compute@minmax@ls{#2}%
2850     \pgfmathsetmacro\pgfmathresult{\forestove{max@s}}%
2851   }%
2852 }
2853 \pgfmathdeclarefunction{max_l}{2}{% #1 = node, #2 = context node (for growth rotation)
2854   \forest@forthis{%
2855     \forest@nameandgo{#1}%
2856     \forest@compute@minmax@ls{#2}%
2857     \pgfmathsetmacro\pgfmathresult{\forestove{max@l}}%
2858   }%
2859 }
2860 \def\forest@compute@minmax@ls#1{% #1 = nodewalk; in the context of which node?
2861   {%
2862     \pgftransformreset
2863     \forest@forthis{%
2864       \forest@nameandgo{#1}%
2865       \pgftransformrotate{-\forestove{grow}}%
2866     }%
2867     \forestoget{min x}\forest@temp@minx
2868     \forestoget{min y}\forest@temp@miny
2869     \forestoget{max x}\forest@temp@maxx
2870     \forestoget{max y}\forest@temp@maxy
2871     \pgfpointransformed{\pgfqpoint{\forest@temp@minx}{\forest@temp@miny}}%
2872     \forestoeset{min@l}{\the\pgf@x}%
2873     \forestoeset{min@s}{\the\pgf@y}%
2874     \forestoeset{max@l}{\the\pgf@x}%
2875     \forestoeset{max@s}{\the\pgf@y}%
2876     \pgfpointransformed{\pgfqpoint{\forest@temp@minx}{\forest@temp@maxy}}%
2877     \ifdim\pgf@x<\forestove{min@l}\relax\forestoeset{min@l}{\the\pgf@x}\fi
2878     \ifdim\pgf@y<\forestove{min@s}\relax\forestoeset{min@s}{\the\pgf@y}\fi
2879     \ifdim\pgf@x>\forestove{max@l}\relax\forestoeset{max@l}{\the\pgf@x}\fi
2880     \ifdim\pgf@y>\forestove{max@s}\relax\forestoeset{max@s}{\the\pgf@y}\fi
2881     \pgfpointransformed{\pgfqpoint{\forest@temp@maxx}{\forest@temp@miny}}%
2882     \ifdim\pgf@x<\forestove{min@l}\relax\forestoeset{min@l}{\the\pgf@x}\fi
2883     \ifdim\pgf@y<\forestove{min@s}\relax\forestoeset{min@s}{\the\pgf@y}\fi
2884     \ifdim\pgf@x>\forestove{max@l}\relax\forestoeset{max@l}{\the\pgf@x}\fi

```

```

2885 \ifdim\pgf@y>\forestove{max@s}\relax\forestoeset{max@s}{\the\pgf@y}\fi
2886 \pgfpointtransformed{\pgfqpoint{\forest@temp@maxx}{\forest@temp@maxy}}}%
2887 \ifdim\pgf@x<\forestove{min@l}\relax\forestoeset{min@l}{\the\pgf@x}\fi
2888 \ifdim\pgf@y<\forestove{min@s}\relax\forestoeset{min@s}{\the\pgf@y}\fi
2889 \ifdim\pgf@x>\forestove{max@l}\relax\forestoeset{max@l}{\the\pgf@x}\fi
2890 \ifdim\pgf@y>\forestove{max@s}\relax\forestoeset{max@s}{\the\pgf@y}\fi
2891 % smuggle out
2892 \edef\forest@marshal{%
2893 \noexpand\forestoeset{min@l}{\forestove{min@l}}}%
2894 \noexpand\forestoeset{min@s}{\forestove{min@s}}}%
2895 \noexpand\forestoeset{max@l}{\forestove{max@l}}}%
2896 \noexpand\forestoeset{max@s}{\forestove{max@s}}}%
2897 }\expandafter
2898 }\forest@marshal
2899 }
2900 \def\forest@pgfmathhelper@attribute@toks#1#2{%
2901 \forest@forthis{%
2902 \forest@nameandgo{#1}%
2903 \ifnum\forest@cn=0
2904 \def\pgfmathresult{%
2905 \else
2906 \forestoget{#2}\pgfmathresult
2907 \fi
2908 }%
2909 }
2910 \def\forest@pgfmathhelper@attribute@dimen#1#2{%
2911 \forest@forthis{%
2912 \forest@nameandgo{#1}%
2913 \ifnum\forest@cn=0
2914 \def\pgfmathresult{Opt}%
2915 \else
2916 \forestoget{#2}\forest@temp
2917 \pgfmathparse{+\forest@temp}%
2918 \fi
2919 }%
2920 }
2921 \def\forest@pgfmathhelper@attribute@count#1#2{%
2922 \forest@forthis{%
2923 \forest@nameandgo{#1}%
2924 \ifnum\forest@cn=0
2925 \def\pgfmathresult{0}%
2926 \else
2927 \forestoget{#2}\forest@temp
2928 \pgfmathtruncatemacro\pgfmathresult{\forest@temp}%
2929 \fi
2930 }%
2931 }
2932 \pgfmathdeclarefunction{id}{1}{%
2933 \forest@forthis{%
2934 \forest@nameandgo{#1}%
2935 \let\pgfmathresult\forest@cn
2936 }%
2937 }
2938 \forestset{%
2939 if id/.code n args={3}{%
2940 \ifnum#1=\forest@cn\relax
2941 \pgfkeysalso{#2}%
2942 \else
2943 \pgfkeysalso{#3}%
2944 \fi
2945 },

```

```

2946   where id/.style n args={3}{for tree={if id={#1}{#2}{#3}}}
2947 }

```

6.5 Nodewalk

Setup machinery.

```

2948 \def\forest@nodewalk@n{0}
2949 \def\forest@nodewalk@historyback{0,}
2950 \def\forest@nodewalk@historyforward{0,}
2951 \def\forest@nodewalk@origin{0}
2952 \def\forest@nodewalk@config@everystep@independent@before#1{% #1 = every step keylist
2953   \foreststrset{every step}{#1}%
2954 }
2955 \def\forest@nodewalk@config@everystep@independent@after{%
2956   \noexpand\foreststrset{every step}{\forestrv{every step}}%
2957 }
2958 \def\forest@nodewalk@config@history@independent@before{%
2959   \def\forest@nodewalk@n{0}%
2960   \edef\forest@nodewalk@origin{\forest@cn}%
2961   \def\forest@nodewalk@historyback{0,}%
2962   \def\forest@nodewalk@historyforward{0,}%
2963 }
2964 \def\forest@nodewalk@config@history@independent@after{%
2965   \edef\noexpand\forest@nodewalk@n{\expandonce{\forest@nodewalk@n}}%
2966   \edef\noexpand\forest@nodewalk@origin{\expandonce{\forest@nodewalk@origin}}%
2967   \edef\noexpand\forest@nodewalk@historyback{\expandonce{\forest@nodewalk@historyback}}%
2968   \edef\noexpand\forest@nodewalk@historyforward{\expandonce{\forest@nodewalk@historyforward}}%
2969 }
2970 \def\forest@nodewalk@config@everystep@shared@before#1{% #1 = every step keylist
2971 \def\forest@nodewalk@config@everystep@shared@after{}
2972 \def\forest@nodewalk@config@history@shared@before{}
2973 \def\forest@nodewalk@config@history@shared@after{}
2974 \def\forest@nodewalk@config@everystep@inherited@before#1{}% #1 = every step keylist
2975 \let\forest@nodewalk@config@everystep@inherited@after\forest@nodewalk@config@everystep@independent@after
2976 \def\forest@nodewalk@config@history@inherited@before{}
2977 \let\forest@nodewalk@config@history@inherited@after\forest@nodewalk@config@history@independent@after
2978 \def\forest@nodewalk#1#2{% #1 = nodewalk, #2 = every step keylist
2979   \def\forest@nodewalk@config@everystep@method{independent}%
2980   \def\forest@nodewalk@config@history@method{independent}%
2981   \def\forest@nodewalk@config@oninvalid{inherited}%
2982   \forest@saveandrestoremacro\forest@nodewalk@oninvalid{%
2983     \forest@nodewalk{#1}{#2}%
2984   }%
2985 }
2986 \def\forest@Nodewalk#1#2{% #1 = nodewalk, #2 = every step keylist
2987   \edef\forest@marshal{%
2988     \noexpand\pgfqkeys{/forest/nodewalk}{\unexpanded{#1}}%
2989     \csname forest@nodewalk@config@everystep\forest@nodewalk@config@everystep@method @after\endcsname
2990     \csname forest@nodewalk@config@history\forest@nodewalk@config@history@method @after\endcsname
2991   }%
2992   \csname forest@nodewalk@config@everystep\forest@nodewalk@config@everystep@method @before\endcsname{#2}%
2993   \csname forest@nodewalk@config@history\forest@nodewalk@config@history@method @before\endcsname
2994   \forest@nodewalk@fakefalse
2995   \forest@marshal
2996 }
2997 \pgfmathdeclarefunction{valid}{1}{%
2998   \forest@forthis{%
2999     \forest@nameandgo{#1}%
3000   \edef\pgfmathresult{\ifnum\forest@cn=0 0\else 1\fi}%
3001   }%
3002 }

```

```

3003 \pgfmathdeclarefunction{invalid}{1}{%
3004   \forest@forthis{%
3005     \forest@nameandgo{#1}%
3006     \edef\pgfmathresult{\ifnum\forest@cn=0 1\else 0\fi}%
3007   }%
3008 }
3009 \newif\ifforest@nodewalk@fake
3010 \def\forest@nodewalk@oninvalid{error}
3011 \def\forest@nodewalk@makestep{%
3012   \ifnum\forest@cn=0
3013     \csname forest@nodewalk@makestep@oninvalid@\forest@nodewalk@config@oninvalid\endcsname
3014   \else
3015     \forest@nodewalk@makestep@
3016   \fi
3017 }
3018 \def\forest@nodewalk@makestep@oninvalid@step{\forest@nodewalk@makestep@}
3019 \def\forest@nodewalk@makestep@oninvalid@error{\PackageError{forest}{nodewalk stepped to the invalid node; sta
3020 \let\forest@nodewalk@makestep@oninvalid@fake\relax
3021 \def\forest@nodewalk@makestep@oninvalid@compatfake{%
3022   \forest@deprecated{last step in stack "\forest@nodewalk@currentstepname", which stepped on an invalid node;
3023 }%
3024 \def\forest@nodewalk@makestep@oninvalid@inherited{\csname forest@nodewalk@makestep@oninvalid@\forest@nodewalk
3025 \def\forest@nodewalk@makestep@{%
3026   \ifforest@nodewalk@fake
3027   \else
3028     \edef\forest@nodewalk@n{\number\numexpr\forest@nodewalk@n+1}%
3029     \epreto\forest@nodewalk@historyback{\forest@cn,}%
3030     \def\forest@nodewalk@historyforward{0,}%
3031     \ifforestdebug\forest@nodewalk@makestep@debug\fi
3032     \forest@process@keylist@register{every step}%
3033   \fi
3034 }
3035 \def\forest@nodewalk@makestep@debug{%
3036   \edef\forest@marshal{%
3037     \noexpand\typeout{\ifforest@nodewalk@fake fake \fi "\forest@nodewalk@currentstepname" step to node id=\fo
3038   }\forest@marshal
3039 }%
3040 \forestset{
3041   debug/.is if=forestdebug,
3042 }
3043 \def\forest@handlers@savecurrentpath{%
3044   \edef\pgfkeyscurrentkey{\pgfkeyscurrentpath}%
3045   \let\forest@currentkey\pgfkeyscurrentkey
3046   \pgfkeys@split@path
3047   \edef\forest@currentpath{\pgfkeyscurrentpath}%
3048   \let\forest@currentname\pgfkeyscurrentname
3049 }
3050 \pgfkeys{/handlers/save current path/.code={\forest@handlers@savecurrentpath}}
3051 \newif\ifforest@nodewalkstephandler@style
3052 \newif\ifforest@nodewalkstephandler@autostep
3053 \newif\ifforest@nodewalkstephandler@stripfakesteps
3054 \newif\ifforest@nodewalkstephandler@muststartatvalidnode
3055 \newif\ifforest@nodewalkstephandler@makefor
3056 \let\forest@nodewalkstephandler@styletrueorfalse\forest@nodewalkstephandler@stylefalse
3057 \def\forest@nodewalk@currentstepname{}
3058 \forestset{
3059   /forest/define@step/style/.is if=forest@nodewalkstephandler@style,
3060   /forest/define@step/autostep/.is if=forest@nodewalkstephandler@autostep,
3061   % the following is useful because some macros use grouping (by \forest@forthis or similar) and therefore, a
3062   /forest/define@step/strip fake steps/.is if=forest@nodewalkstephandler@stripfakesteps,
3063   % this can never happen with autosteps ...

```



```

3064 /forest/define@step/autostep/.append code={%
3065   \ifforest@nodewalkstephandler@autostep
3066     \forest@nodewalkstephandler@stripfakestepsfalse
3067   \fi
3068 },
3069 /forest/define@step/must start at valid node/.is if=forest@nodewalkstephandler@muststartatvalidnode,
3070 /forest/define@step/n args/.store in=\forest@nodewalkstephandler@nargs,
3071 /forest/define@step/make for/.is if=forest@nodewalkstephandler@makefor,
3072 /forest/define@step/@bare/.style={strip fake steps=false,must start at valid node=false,make for=false},
3073 define long step/.code n args=3{%
3074   \forest@nodewalkstephandler@styletrueorfalse % true for end users; but in the package, most of steps are
3075   \forest@nodewalkstephandler@autostepfalse
3076   \forest@nodewalkstephandler@stripfakesteptrue
3077   \forest@nodewalkstephandler@muststartatvalidnodetrue % most steps can only start at a valid node
3078   \forest@nodewalkstephandler@makefortrue % make for prefix?
3079   \def\forest@nodewalkstephandler@nargs{0}%
3080   \pgfqkeys{/forest/define@step}{#2}%
3081   \forest@temp@toks{#3}% handler code
3082   \ifforest@nodewalkstephandler@style
3083     \expandafter\forest@temp@toks\expandafter{%
3084       \expandafter\pgfqkeysalso\expandafter{\the\forest@temp@toks}%
3085     }%
3086   \fi
3087   \ifforest@nodewalkstephandler@autostep
3088     \apptotoks\forest@temp@toks{\forest@nodewalk@makestep}%
3089   \fi
3090   \ifforest@nodewalkstephandler@stripfakesteps
3091     \expandafter\forest@temp@toks\expandafter{\expandafter\forest@nodewalk@stripfakesteps\expandafter{\the\
3092   \fi
3093   \ifforest@nodewalkstephandler@muststartatvalidnode
3094     \edef\forest@marshal{%
3095       \noexpand\forest@temp@toks{%
3096         \unexpanded{%
3097           \ifnum\forest@cn=0
3098             \csname forest@nodewalk@start@oninvalid@\forest@nodewalk@oninvalid\endcsname{#1}%
3099           \else
3100             }%
3101           \noexpand\@escapeif{\the\forest@temp@toks}%
3102           \noexpand\fi
3103         }%
3104       }\forest@marshal
3105     \fi
3106     \pretotoks\forest@temp@toks{\appto\forest@nodewalk@currentstepname{,#1}}%
3107     \expandafter\forest@temp@toks\expandafter{\expandafter\forest@saveandrestoremacro\expandafter\forest@node
3108   \ifforestdebug
3109     \epretotoks\forest@temp@toks{\noexpand\typeout{Starting step "#1" from id=\forest@cn
3110       \ifnum\forest@nodewalkstephandler@nargs>0 with args #####1\fi
3111       \ifnum\forest@nodewalkstephandler@nargs>1 ,####2\fi
3112       \ifnum\forest@nodewalkstephandler@nargs>2 ,####3\fi
3113       \ifnum\forest@nodewalkstephandler@nargs>3 ,####4\fi
3114       \ifnum\forest@nodewalkstephandler@nargs>4 ,####5\fi
3115       \ifnum\forest@nodewalkstephandler@nargs>5 ,####6\fi
3116       \ifnum\forest@nodewalkstephandler@nargs>6 ,####7\fi
3117       \ifnum\forest@nodewalkstephandler@nargs>7 ,####8\fi
3118       \ifnum\forest@nodewalkstephandler@nargs>8 ,####9\fi
3119     }}%
3120   \fi
3121   \def\forest@temp[/forest/nodewalk/#1/.code]%
3122   \ifnum\forest@nodewalkstephandler@nargs<2
3123     \appto\forest@temp{=}%
3124   \else\ifnum\forest@nodewalkstephandler@nargs=2

```

```

3125     \eappto\forest@temp{ 2 args=}%
3126 \else
3127     \eappto\forest@temp{ n args={\forest@nodewalkstephandler@nargs}}%
3128 \fi\fi
3129 \eappto\forest@temp{{\the\forest@temp@toks}}%
3130 \expandafter\pgfkeysalso\expandafter{\forest@temp}%
3131 \ifforest@nodewalkstephandler@makefor
3132     \ifnum\forest@nodewalkstephandler@nargs=0
3133         \forestset{%
3134             for #1/.code={\forest@forstepwrapper{#1}{##1}},
3135         }%
3136 \else\ifnum\forest@nodewalkstephandler@nargs=1
3137     \forestset{%
3138         for #1/.code 2 args={\forest@forstepwrapper{#1={##1}}{##2}},
3139     }%
3140 \else
3141     \forestset{%
3142         for #1/.code n args/.expanded=%
3143             {\number\numexpr\forest@nodewalkstephandler@nargs+1}%
3144             {\noexpand\forest@forstepwrapper{#1\ifnum\forest@nodewalkstephandler@nargs>0=\fi\forest@util@narg
3145         }}%
3146     \fi\fi
3147 \fi
3148 },
3149 }

```

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

```

3150 \def\forest@forstepwrapper#1#2{\forest@forthis{\forest@nodewalk{#1}{#2}}}
3151 \def\forest@util@nargs#1#2#3{% #1 = prefix (#, ##, ...), #2 = n args, #3=start; returns {#start}{#start+1}...
3152     \ifnum#2>0 {#1\number\numexpr#3+1}\fi
3153     \ifnum#2>1 {#1\number\numexpr#3+2}\fi
3154     \ifnum#2>2 {#1\number\numexpr#3+3}\fi
3155     \ifnum#2>3 {#1\number\numexpr#3+4}\fi
3156     \ifnum#2>4 {#1\number\numexpr#3+5}\fi
3157     \ifnum#2>5 {#1\number\numexpr#3+6}\fi
3158     \ifnum#2>6 {#1\number\numexpr#3+7}\fi
3159     \ifnum#2>7 {#1\number\numexpr#3+8}\fi
3160     \ifnum#2>8 {#1\number\numexpr#3+9}\fi
3161 }
3162 \def\forest@nodewalk@start@oninvalid@fake#1{}
3163 \def\forest@nodewalk@start@oninvalid@real#1{}
3164 \def\forest@nodewalk@start@oninvalid@error#1{\PackageError{forest}{nodewalk step "#1" cannot start at the inv

```

Define long-form single-step walks.

```

3165 \forestset{
3166     define long step={current}{autostep}{},
3167     define long step={next}{autostep}{\edef\forest@cn{\forestove{@next}}},
3168     define long step={previous}{autostep}{\edef\forest@cn{\forestove{@previous}}},
3169     define long step={parent}{autostep}{\edef\forest@cn{\forestove{@parent}}},
3170     define long step={first}{autostep}{\edef\forest@cn{\forestove{@first}}},
3171     define long step={last}{autostep}{\edef\forest@cn{\forestove{@last}}},
3172     define long step={sibling}{autostep}{%
3173         \edef\forest@cn{%
3174             \ifnum\forestove{@previous}=0
3175                 \forestove{@next}%
3176             \else
3177                 \forestove{@previous}%
3178             \fi
3179         }%
3180 },

```

```

3181 define long step={next node}{autostep}{\edef\forest@cn{\forest@node@linearnextid}},
3182 define long step={previous node}{autostep}{\edef\forest@cn{\forest@node@linearpreviousid}},
3183 define long step={first leaf}{autostep}{%
3184   \safeloop
3185   \edef\forest@cn{\forestove{@first}}%
3186   \unless\ifnum\forestove{@first}=0
3187   \saferepeat
3188 },
3189 define long step={first leaf'}{autostep}{%
3190   \safeloop
3191   \unless\ifnum\forestove{@first}=0
3192   \edef\forest@cn{\forestove{@first}}%
3193   \saferepeat
3194 },
3195 define long step={last leaf}{autostep}{%
3196   \safeloop
3197   \edef\forest@cn{\forestove{@last}}%
3198   \unless\ifnum\forestove{@last}=0
3199   \saferepeat
3200 },
3201 define long step={last leaf'}{autostep}{%
3202   \safeloop
3203   \unless\ifnum\forestove{@last}=0
3204   \edef\forest@cn{\forestove{@last}}%
3205   \saferepeat
3206 },
3207 define long step={next leaf}{style,strip fake steps=false}{group={do until={n_children()==0}{next node}}},
3208 define long step={previous leaf}{style,strip fake steps=false}{group={do until={n_children()==0}{previous node}}},
3209 define long step={next on tier}{autostep}{%
3210   \def\forest@temp{#1}%
3211   \ifx\forest@temp\pgfkeysnovalue@text
3212     \foresttoget{tier}\forest@nodewalk@giventier
3213   \else
3214     \def\forest@nodewalk@giventier{#1}%
3215   \fi
3216   \edef\forest@cn{\forest@node@linearnextnotdescendantid}%
3217   \safeloop
3218   \foresttoget{tier}\forest@nodewalk@tier
3219   \unless\ifx\forest@nodewalk@tier\forest@nodewalk@giventier
3220     \edef\forest@cn{\forest@node@linearnextid}%
3221   \saferepeat
3222 },
3223 define long step={previous on tier}{autostep}{%
3224   \def\forest@temp{#1}%
3225   \ifx\forest@temp\pgfkeysnovalue@text
3226     \foresttoget{tier}\forest@nodewalk@giventier
3227   \else
3228     \def\forest@nodewalk@giventier{#1}%
3229   \fi
3230   \safeloop
3231   \edef\forest@cn{\forest@node@linearpreviousid}%
3232   \foresttoget{tier}\forest@nodewalk@tier
3233   \unless\ifx\forest@nodewalk@tier\forest@nodewalk@giventier
3234     \saferepeat
3235 },
3236 %
3237 define long step={root}{autostep,must start at valid node=false}{%
3238   \edef\forest@cn{\forest@node@rootid}},
3239 define long step={root'}{autostep,must start at valid node=false}{%
3240   \forest0ifdefined{\forest@root}{@parent}{\edef\forest@cn{\forest@root}}{\edef\forest@cn{0}}%
3241 },

```

```

3242 define long step={origin}{autostep,must start at valid node=false}{\edef\forest@cn{\forest@nodewalk@origin}%
3243 %
3244 define long step={n}{autostep,n args=1}{%
3245   \pgfmathtruncatemacro\forest@temp{n}{#1}%
3246   \edef\forest@cn{\forest@node@nthchildid{\forest@temp{n}}}%
3247 },
3248 define long step={n}{autostep,make for=false,n args=1}{%
3249   % Yes, twice. ;-)
3250   % n=1 and n(ext)
3251   \def\forest@nodewalk@temp{#1}%
3252   \ifx\forest@nodewalk@temp\pgfkeysnovalue@text
3253     \edef\forest@cn{\forest@node@next}%
3254   \else
3255     \pgfmathtruncatemacro\forest@temp{n}{#1}%
3256     \edef\forest@cn{\forest@node@nthchildid{\forest@temp{n}}}%
3257   \fi
3258 },
3259 define long step={n'}{autostep,n args=1}{%
3260   \pgfmathtruncatemacro\forest@temp{n}{#1}%
3261   \edef\forest@cn{\forest@node@nbarthchildid{\forest@temp{n}}}%
3262 },
3263 define long step={to tier}{autostep,n args=1}{%
3264   \def\forest@nodewalk@giventier{#1}%
3265   \safeloop
3266     \forest@toget{tier}\forest@nodewalk@tier
3267   \unless\ifx\forest@nodewalk@tier\forest@nodewalk@giventier
3268     \forest@toget{@parent}\forest@cn
3269   \safererepeat
3270 },
3271 %
3272 define long step={name}{autostep,n args=1,must start at valid node=false}{%
3273   \edef\forest@cn{%
3274     \forest@node@ifnamedefined{#1}{\forest@node@Nametoid{#1}}{0}%
3275   }%
3276 },
3277 define long step={id}{autostep,n args=1,must start at valid node=false}{%
3278   \forest@ifdefined{#1}{@parent}{\edef\forest@cn{#1}}{\edef\forest@cn{0}}%
3279 },
3280 define long step={Nodewalk}{n args=3,@bare}{% #1 = config, #2 = nodewalk
3281   \def\forest@nodewalk@config@everystep@method{independent}%
3282   \def\forest@nodewalk@config@history@method{shared}%
3283   \def\forest@nodewalk@config@oninvalid{inherited}%
3284   \forest@saveandrestoremacro\forest@nodewalk@oninvalid{%
3285     \pgfkeys{/forest/nodewalk@config}{#1}%
3286     \forest@Nodewalk{#2}{#3}%
3287   }%
3288 },
3289 define long step={nodewalk}{n args=2,@bare}{% #1 = nodewalk, #2 = every step
3290   \forest@nodewalk{#1}{#2}%
3291 },
3292 define long step={nodewalk'}{n args=1,@bare}{% #1 = nodewalk, #2 = every step
3293   \def\forest@nodewalk@config@everystep@method{inherited}%
3294   \def\forest@nodewalk@config@history@method{independent}%
3295   \def\forest@nodewalk@config@oninvalid{inherited}%
3296   \forest@saveandrestoremacro\forest@nodewalk@oninvalid{%
3297     \forest@Nodewalk{#1}{}%
3298   }%
3299 },
3300 % these must be defined explicitly, as prefix "for" normally introduces the every-step keylist
3301 for nodewalk/.code 2 args={%
3302   \forest@forthis{\forest@nodewalk{#1}{#2}}},

```

```

3303 for nodewalk'/.code={%
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     \forest@forthis{\forest@Nodewalk{#1}{}}}%
3309   }%
3310 },
3311 for Nodewalk/.code n args=3{% #1 = config, #2 = nodewalk, #3 = every-step
3312   \def\forest@nodewalk@config@everystep@method{inherited}%
3313   \def\forest@nodewalk@config@history@method{independent}%
3314   \def\forest@nodewalk@config@oninvalid{inherited}%
3315   \forest@saveandrestoremacro\forest@nodewalk@oninvalid{%
3316     \pgfqkeys{/forest/nodewalk@config}{#1}%
3317     \forest@forthis{\forest@Nodewalk{#2}{#3}}}%
3318   }%
3319 },
3320 copy command key={/forest/for nodewalk}{/forest/nodewalk/for nodewalk},
3321 copy command key={/forest/for nodewalk'}{/forest/nodewalk/for nodewalk'},
3322 copy command key={/forest/for Nodewalk}{/forest/nodewalk/for Nodewalk},
3323 declare keylist register=every step,
3324 every step'={},
3325 %%% begin nodewalk config
3326 nodewalk@config/.cd,
3327 every@step/.is choice,
3328 every@step/independent/.code={},
3329 every@step/inherited/.code={},
3330 every@step/shared/.code={},
3331 every step/.store in=\forest@nodewalk@config@everystep@method,
3332 every step/.prefix style={every@step=#1},
3333 @history/.is choice,
3334 @history/independent/.code={},
3335 @history/inherited/.code={},
3336 @history/shared/.code={},
3337 history/.store in=\forest@nodewalk@config@history@method,
3338 history/.prefix style={@history=#1},
3339 on@invalid/.is choice,
3340 on@invalid/error/.code={},
3341 on@invalid/fake/.code={},
3342 on@invalid/step/.code={},
3343 on@invalid/inherited/.code={},
3344 on invalid/.store in=\forest@nodewalk@config@oninvalid,
3345 on invalid/.prefix style={on@invalid=#1},
3346 %%% end nodewalk config
3347 }
3348 \forestset{
3349   define long step={branch}{n args=1,@bare,style}{%
3350     branch@@build/.style={branch@build={##1}{}},
3351     @branch={#1},
3352   },
3353   define long step={branch'}{n args=1,@bare,style}{%
3354     branch@@build/.style/.expanded={branch@build={###1}{\forestregister{every step},}},
3355     @branch={#1},
3356   },
3357   nodewalk/@branch/.style={
3358     TeX={\forest@temp@toks{}},
3359     split/.process args={r}{#1}{,}{branch@@build},
3360     branch@do,
3361   },
3362   nodewalk/branch@build/.code 2 args={% #1 = nodewalk for this branch, #2 = perhaps every step keylist
3363     \ifstrempy{#1}{}%

```

```

3364 \expandafter\ifstremp\expandafter{\the\forest@temp@toks}{%
3365 \edef\forest@marshal{%
3366 \noexpand\forest@temp@toks{for nodewalk={\unexpanded{#1}}{\forestregister{every step}}}%
3367 }\forest@marshal
3368 }{%
3369 \edef\forest@marshal{%
3370 \noexpand\forest@temp@toks{for nodewalk={\unexpanded{#1}}{#2\the\forest@temp@toks}}%
3371 }\forest@marshal
3372 }%
3373 }%
3374 },
3375 nodewalk/branch@do/.code={%
3376 \edef\forest@marshal{%
3377 \noexpand\pgfkeysalso{fake={\the\forest@temp@toks}}%
3378 }\forest@marshal
3379 },
3380 define long step={group}{autostep}{\forest@go{#1}},
3381 nodewalk/fake/.code={%
3382 \forest@saveandrestoreifcs{forest@nodewalk@fake}{%
3383 \forest@nodewalk@faketrue
3384 \pgfkeysalso{#1}%
3385 }%
3386 },
3387 nodewalk/real/.code={%
3388 \forest@saveandrestoreifcs{forest@nodewalk@fake}{%
3389 \forest@nodewalk@fakefalse
3390 \pgfkeysalso{#1}%
3391 }%
3392 },
3393 define long step={filter}{n args=2,@bare,style}{% #1 = nodewalk, #2 = condition
3394 nodewalk/.expanded={\unexpanded{#1}}{if={\unexpanded{#2}}{\forestregister{every step}}{}}{}}
3395 },
3396 on@invalid/.is choice,
3397 on@invalid/error/.code={},
3398 on@invalid/fake/.code={},
3399 on@invalid/step/.code={},
3400 on invalid/.code 2 args={%
3401 \pgfkeysalso{/forest/on@invalid={#1}}%
3402 \forest@saveandrestoremacro\forest@nodewalk@oninvalid{%
3403 \def\forest@nodewalk@oninvalid{#1}%
3404 \pgfkeysalso{#2}%
3405 }%
3406 },
3407 define long step={strip fake steps}{n args=1,@bare}{%
3408 \forest@nodewalk@stripfakesteps{\pgfkeysalso{#1}}},
3409 define long step={walk back}{n args=1,@bare}{%
3410 \pgfmathtruncatemacro\forest@temp@n{#1}%
3411 \forest@nodewalk@walklist{\forest@nodewalk@historyforward}{\forest@nodewalk@historyback}{\ifnum\forest@cn
3412 \forest@nodewalk@back@updatehistory
3413 },
3414 nodewalk/walk back/.default=1,
3415 define long step={jump back}{n args=1,@bare}{%
3416 \pgfmathtruncatemacro\forest@temp@n{(#1)+\ifnum\forest@cn=0 0\else1\fi}%
3417 \forest@nodewalk@walklist{\forest@nodewalk@historyforward}{\forest@nodewalk@historyback}{\forest@temp@n-1
3418 \forest@nodewalk@back@updatehistory
3419 },
3420 nodewalk/jump back/.default=1,
3421 define long step={back}{n args=1,@bare}{%
3422 \pgfmathtruncatemacro\forest@temp@n{#1}%
3423 \forest@nodewalk@walklist{\forest@nodewalk@historyforward}{\forest@nodewalk@historyback}{\ifnum\forest@cn
3424 \forest@nodewalk@back@updatehistory

```

```

3425 },
3426 nodewalk/back/.default=1,
3427 define long step={walk forward}{n args=1,@bare}{%
3428   \pgfmathtruncatemacro\forest@temp@n{#1}%
3429   \forest@nodewalk@walklist{\forest@nodewalk@historyback}{\forest@nodewalk@historyforward}{0}{\forest@temp@n-1}
3430   \forest@nodewalk@forward@updatehistory
3431 },
3432 nodewalk/walk forward/.default=1,
3433 define long step={jump forward}{n args=1,@bare}{%
3434   \pgfmathtruncatemacro\forest@temp@n{#1}%
3435   \forest@nodewalk@walklist{\forest@nodewalk@historyback}{\forest@nodewalk@historyforward}{\forest@temp@n-1}
3436   \forest@nodewalk@forward@updatehistory
3437 },
3438 nodewalk/jump forward/.default=1,
3439 define long step={forward}{n args=1,@bare}{%
3440   \pgfmathtruncatemacro\forest@temp@n{#1}%
3441   \forest@nodewalk@walklist{\forest@nodewalk@historyback}{\forest@nodewalk@historyforward}{0}{\forest@temp@n-1}
3442   \forest@nodewalk@forward@updatehistory
3443 },
3444 nodewalk/forward/.default=1,
3445 define long step={last valid'}{@bare}{%
3446   \ifnum\forest@cn=0
3447     \forest@nodewalk@tolastvalid
3448     \forest@nodewalk@makestep
3449   \fi
3450 },
3451 define long step={last valid}{@bare}{%
3452   \forest@nodewalk@tolastvalid
3453 },
3454 define long step={reverse}{n args=1,@bare,make for}{%
3455   \forest@nodewalk{#1,TeX={%
3456     \global\let\forest@global@temp\forest@nodewalk@historyback
3457     \global\let\forest@global@tempn\forest@nodewalk@n
3458   }}{%
3459     \forest@nodewalk@walklist{\forest@global@temp}{0}{\forest@global@tempn}{\let\forest@cn\forest@nodewalk@n}
3460   },
3461   define long step={walk and reverse}{n args=1,@bare,make for}{%
3462     \edef\forest@marshal{%
3463       \noexpand\pgfkeysalso{\unexpanded{#1}}%
3464       \noexpand\forest@nodewalk@walklist{\forest@nodewalk@historyback}{0}{\noexpand\forest@nodewalk@n}
3465     }\forest@marshal
3466   },
3467   define long step={sort}{n args=1,@bare,make for}{%
3468     \forest@nodewalk{#1,TeX={%
3469       \global\let\forest@global@temp\forest@nodewalk@historyback
3470       \global\let\forest@global@tempn\forest@nodewalk@n
3471     }}{%
3472       \forest@nodewalk@sortlist{\forest@global@temp}{\forest@global@tempn}\forest@sort@ascending
3473     },
3474     define long step={sort'}{n args=1,@bare,make for}{%
3475       \forest@nodewalk{#1,TeX={%
3476         \global\let\forest@global@temp\forest@nodewalk@historyback
3477         \global\let\forest@global@tempn\forest@nodewalk@n
3478       }}{%
3479         \forest@nodewalk@sortlist{\forest@global@temp}{\forest@global@tempn}\forest@sort@descending
3480       },
3481       define long step={walk and sort}{n args=1,@bare,make for}{% walk as given, then walk sorted
3482         \edef\forest@marshal{%
3483           \noexpand\pgfkeysalso{\unexpanded{#1}}%
3484           \noexpand\forest@nodewalk@sortlist{\noexpand\forest@nodewalk@historyback}{\noexpand\forest@nodewalk@n-1}
3485         }\forest@marshal

```

```

3486 },
3487 define long step={walk and sort'}{n args=1,@bare,make for}{%
3488   \edef\forest@marshal{%
3489     \noexpand\pgfkeysalso{\unexpanded{#1}}%
3490     \noexpand\forest@nodewalk@sortlist{\noexpand\forest@nodewalk@historyback}{\noexpand\forest@nodewalk@n-}
3491   }\forest@marshal
3492 },
3493 sort by/.store in=\forest@nodesort@by,
3494 define long step={save}{n args=2,@bare,make for}{% #1 = name, #2 = nodewalk
3495   \forest@forthis{%
3496     \forest@nodewalk{#2,TeX={%
3497       \global\let\forest@global@temp\forest@nodewalk@historyback
3498       \global\let\forest@global@tempn\forest@nodewalk@n
3499     }}{%
3500   }%
3501   \forest@nodewalk@walklist{}{\forest@global@temp}{0}{\forest@global@tempn}\relax
3502   \csedef{forest@nodewalk@saved@#1}{\forest@nodewalk@walklist@walked}%
3503 },
3504 define long step={walk and save}{n args=2,@bare,make for}{% #1 = name, #2 = nodewalk
3505   \edef\forest@marshal{%
3506     \noexpand\pgfkeysalso{\unexpanded{#2}}%
3507     \noexpand\forest@nodewalk@walklist{}{\noexpand\forest@nodewalk@historyback}{0}{\noexpand\forest@nodewalk@n-}
3508   }\forest@marshal
3509   \csedef{forest@nodewalk@saved@#1}{\forest@nodewalk@walklist@walked}%
3510 },
3511 nodewalk/save history/.code 2 args={% #1 = back, forward
3512   \csedef{forest@nodewalk@saved@#1}{\forest@nodewalk@historyback}%
3513   \csedef{forest@nodewalk@saved@#2}{\forest@nodewalk@historyforward}%
3514 },
3515 define long step={load}{n args=1,@bare,make for}{%
3516   \forest@nodewalk@walklist{}{\csuse{forest@nodewalk@saved@#1}0,}{0}{-1}{\ifnum\forest@nodewalk@cn=0 \else\
3517 },
3518 if in saved nodewalk/.code n args=4{% is node #1 in nodewalk #2; yes: #3, no: #4
3519   \forest@forthis{%
3520     \forest@go{#1}%
3521     \edef\forest@marshal{%
3522       \noexpand\pgfutil@in@{,\forest@cn,}{,\csuse{forest@nodewalk@saved@#2},}%
3523     }\forest@marshal
3524   }%
3525   \ifpgfutil@in@
3526     \@escapeif{\pgfkeysalso{#3}}%
3527   \else
3528     \@escapeif{\pgfkeysalso{#4}}%
3529   \fi
3530 },
3531 where in saved nodewalk/.style n args=4{
3532   for tree={if in saved nodewalk={#1}{#2}{#3}{#4}}
3533 },
3534 nodewalk/options/.code={\forestset{#1}},
3535 nodewalk/TeX/.code={#1},
3536 nodewalk/TeX'/.code={\appto\forest@externalize@loadimages{#1}{#1},
3537 nodewalk/TeX''/.code={\appto\forest@externalize@loadimages{#1}},
3538 nodewalk/typeout/.style={TeX={\typeout{#1}}},
3539 % repeat is taken later from /forest/repeat
3540 }
3541 \def\forest@nodewalk@walklist#1#2#3#4#5{%
3542   % #1 = list of preceding, #2 = list to walk
3543   % #3 = from, #4 = to
3544   % #5 = every step code
3545   \let\forest@nodewalk@cn\forest@cn
3546   \edef\forest@marshal{%

```



```

3547 \noexpand\forest@nodewalk@walklist@{#1}-{#2}{\number\numexpr#3}{\number\numexpr#4}{1}{0}{\unexpanded{#5}}}%
3548 }\forest@marshal
3549 }
3550 \def\forest@nodewalk@walklist@#1#2#3#4#5#6#7{%
3551 % #1 = list of walked, #2 = list to walk
3552 % #3 = from, #4 = to
3553 % #5 = current step n, #6 = steps made
3554 % #7 = every step code
3555 \def\forest@nodewalk@walklist@walked{#1}%
3556 \def\forest@nodewalk@walklist@rest{#2}%
3557 \edef\forest@nodewalk@walklist@stepsmade{#6}%
3558 \ifnum#4<0
3559 \forest@temptrue
3560 \else
3561 \ifnum#5>#4\relax
3562 \forest@tempfalse
3563 \else
3564 \forest@temptrue
3565 \fi
3566 \fi
3567 \ifforest@temp
3568 \edef\forest@nodewalk@cn{\forest@csvlist@getfirst@{#2}}%
3569 \ifnum\forest@nodewalk@cn=0
3570 #7%
3571 \else
3572 \ifnum#5>#3\relax
3573 #7%
3574 \edef\forest@nodewalk@walklist@stepsmade{\number\numexpr#6+1}%
3575 \fi
3576 \forest@csvlist@getfirstrest@{#2}\forest@nodewalk@cn\forest@nodewalk@walklist@rest
3577 \@escapeiffif{%
3578 \edef\forest@marshal{%
3579 \noexpand\forest@nodewalk@walklist@
3580 {\forest@nodewalk@cn,#1}{\forest@nodewalk@walklist@rest}{#3}{#4}{\number\numexpr#5+1}{\forest@nod
3581 }\forest@marshal
3582 }%
3583 \fi
3584 \fi
3585 }
3586
3587 \def\forest@nodewalk@back@updatehistory{%
3588 \ifnum\forest@cn=0
3589 \let\forest@nodewalk@historyback\forest@nodewalk@walklist@rest
3590 \let\forest@nodewalk@historyforward\forest@nodewalk@walklist@walked
3591 \else
3592 \expandafter\forest@csvlist@getfirstrest@\expandafter{\forest@nodewalk@walklist@walked}\forest@temp\fores
3593 \edef\forest@nodewalk@historyback{\forest@temp,\forest@nodewalk@walklist@rest}%
3594 \fi
3595 }
3596 \def\forest@nodewalk@forward@updatehistory{%
3597 \let\forest@nodewalk@historyforward\forest@nodewalk@walklist@rest
3598 \let\forest@nodewalk@historyback\forest@nodewalk@walklist@walked
3599 }
3600 \def\forest@go#1{%
3601 \def\forest@nodewalk@config@everystep@method{independent}%
3602 \def\forest@nodewalk@config@history@method{inherited}%
3603 \def\forest@nodewalk@config@oninvalid{inherited}%
3604 \forest@saveandrestoremacro\forest@nodewalk@oninvalid{%
3605 \forest@Nodewalk{#1}{}%
3606 }%
3607 }

```

```

3608 \def\forest@csvlist@getfirst@#1{% assuming that the list is nonempty and finishes with a comma
3609 \forest@csvlist@getfirst@@#1\forest@csvlist@getfirst@@}
3610 \def\forest@csvlist@getfirst@@#1,#2\forest@csvlist@getfirst@@{#1}
3611 \def\forest@csvlist@getrest@#1{% assuming that the list is nonempty and finishes with a comma
3612 \forest@csvlist@getrest@@#1\forest@csvlist@getrest@@}
3613 \def\forest@csvlist@getrest@@#1,#2\forest@csvlist@getrest@@{#2}
3614 \def\forest@csvlist@getfirstrest@#1#2#3{% assuming that the list is nonempty and finishes with a comma
3615 % #1 = list, #2 = cs receiving first, #3 = cs receiving rest
3616 \forest@csvlist@getfirstrest@@#1\forest@csvlist@getfirstrest@@{#2}{#3}}
3617 \def\forest@csvlist@getfirstrest@@#1,#2\forest@csvlist@getfirstrest@@#3#4{%
3618 \def#3{#1}%
3619 \def#4{#2}%
3620 }
3621 \def\forest@nodewalk@stripfakesteps#1{%
3622 % go to the last valid node if the walk contained any nodes, otherwise restore the current node
3623 \edef\forest@marshal{%
3624 \unexpanded{#1}%
3625 \noexpand\ifnum\noexpand\forest@nodewalk@n=\forest@nodewalk@n\relax
3626 \def\noexpand\forest@cn{\forest@cn}%
3627 \noexpand\else
3628 \unexpanded{%
3629 \edef\forest@cn{%
3630 \expandafter\forest@csvlist@getfirst@\expandafter{\forest@nodewalk@historyback}%
3631 }%
3632 }%
3633 \noexpand\fi
3634 }\forest@marshal
3635 }
3636 \def\forest@nodewalk@tolastvalid{%
3637 \ifnum\forest@cn=0
3638 \edef\forest@cn{\expandafter\forest@csvlist@getfirst@\expandafter{\forest@nodewalk@historyback}}%
3639 \ifnum\forest@cn=0
3640 \let\forest@cn\forest@nodewalk@origin
3641 \fi
3642 \fi
3643 }
3644 \def\forest@nodewalk@sortlist#1#2#3{%#1=list,#2=to,#3=asc/desc
3645 \edef\forest@nodewalksort@list{#1}%
3646 \expandafter\forest@nodewalk@sortlist@\expandafter{\number\numexpr#2}{#3}%
3647 }
3648 \def\forest@nodewalk@sortlist@#1#2{%#1=to,#2=asc/desc
3649 \safeloop
3650 \unless\ifnum\safeloopn>#1\relax
3651 \expandafter\forest@csvlist@getfirstrest@\expandafter{\forest@nodewalksort@list}\forest@nodewalksort@cn\fi
3652 \csedef\forest@nodesort@\safeloopn{\forest@nodewalksort@cn}%
3653 \saferepeat
3654 \edef\forest@nodesort@sortkey{\forest@nodesort@by}%
3655 \forest@sort\forest@nodesort@cmpnodes\forest@nodesort@let#2{1}{#1}%
3656 \def\forest@nodewalksort@sorted{}%
3657 \safeloop
3658 \unless\ifnum\safeloopn>#1\relax
3659 \edef\forest@cn{\csname forest@nodesort@\safeloopn\endcsname}%
3660 \forest@nodewalk@makestep
3661 \saferepeat
3662 }

```

Find minimal/maximal node in a walk.

```

3663 \forestset{
3664   define long step={min}{n args=1,@bare,make for}{% the first min in the argument nodewalk
3665     \forest@nodewalk{#1,TeX={%
3666       \global\let\forest@global@temp\forest@nodewalk@historyback

```

```

3667     }}{}%
3668     \forest@nodewalk@minmax{\forest@global@temp}{-1}{<}{\forest@nodewalk@minmax@node,}%
3669 },
3670 define long step={mins}{n args=1,@bare,make for}{% all mins in the argument nodewalk
3671     \forest@nodewalk{#1,TeX={%
3672         \global\let\forest@global@temp\forest@nodewalk@historyback
3673     }}{}%
3674     \forest@nodewalk@minmax{\forest@global@temp}{-1}{<}{\forest@nodewalk@minmax@nodes}%
3675 },
3676 define long step={walk and min}{n args=1,@bare}{%
3677     \edef\forest@marshal{%
3678         \noexpand\pgfkeysalso{\unexpanded{#1}}%
3679         \noexpand\forest@nodewalk@minmax{\noexpand\forest@nodewalk@historyback}{\noexpand\forest@nodewalk@n-\fo
3680     }\forest@marshal
3681 },
3682 define long step={walk and mins}{n args=1,@bare}{%
3683     \edef\forest@marshal{%
3684         \noexpand\pgfkeysalso{\unexpanded{#1}}%
3685         \noexpand\forest@nodewalk@minmax{\noexpand\forest@nodewalk@historyback}{\noexpand\forest@nodewalk@n-\fo
3686     }\forest@marshal
3687 },
3688 define long step={min in nodewalk}{@bare}{% find the first min in the preceding nodewalk, step to it
3689     \forest@nodewalk@minmax{\forest@nodewalk@historyback}{-1}{<}{\forest@nodewalk@minmax@node,}%
3690 },
3691 define long step={mins in nodewalk}{@bare}{% find mins in the preceding nodewalk, step to mins
3692     \forest@nodewalk@minmax{\forest@nodewalk@historyback}{-1}{<}{\forest@nodewalk@minmax@nodes}%
3693 },
3694 define long step={min in nodewalk'}{@bare}{% find the first min in the preceding nodewalk, step to min in h
3695     \forest@nodewalk@minmax{\forest@nodewalk@historyback}{-1}{<}{}%
3696 },
3697 %
3698 define long step={max}{n args=1,@bare,make for}{% the first max in the argument nodewalk
3699     \forest@nodewalk{#1,TeX={%
3700         \global\let\forest@global@temp\forest@nodewalk@historyback
3701     }}{}%
3702     \forest@nodewalk@minmax{\forest@global@temp}{-1}{>}{\forest@nodewalk@minmax@node,}%
3703 },
3704 define long step={maxs}{n args=1,@bare,make for}{% all maxs in the argument nodewalk
3705     \forest@nodewalk{#1,TeX={%
3706         \global\let\forest@global@temp\forest@nodewalk@historyback
3707     }}{}%
3708     \forest@nodewalk@minmax{\forest@global@temp}{-1}{>}{\forest@nodewalk@minmax@nodes}%
3709 },
3710 define long step={walk and max}{n args=1,@bare}{%
3711     \edef\forest@marshal{%
3712         \noexpand\pgfkeysalso{\unexpanded{#1}}%
3713         \noexpand\forest@nodewalk@minmax{\noexpand\forest@nodewalk@historyback}{\noexpand\forest@nodewalk@n-\fo
3714     }\forest@marshal
3715 },
3716 define long step={walk and maxs}{n args=1,@bare}{%
3717     \edef\forest@marshal{%
3718         \noexpand\pgfkeysalso{\unexpanded{#1}}%
3719         \noexpand\forest@nodewalk@minmax{\noexpand\forest@nodewalk@historyback}{\noexpand\forest@nodewalk@n-\fo
3720     }\forest@marshal
3721 },
3722 define long step={max in nodewalk}{@bare}{% find the first max in the preceding nodewalk, step to it
3723     \forest@nodewalk@minmax{\forest@nodewalk@historyback}{-1}{>}{\forest@nodewalk@minmax@node,}%
3724 },
3725 define long step={maxs in nodewalk}{@bare}{% find maxs in the preceding nodewalk, step to maxs
3726     \forest@nodewalk@minmax{\forest@nodewalk@historyback}{-1}{>}{\forest@nodewalk@minmax@nodes}%
3727 },

```

```

3728   define long step={max in nodewalk'}{@bare}{% find the first max in the preceding nodewalk, step to max in h
3729     \forest@nodewalk@minmax{\forest@nodewalk@historyback}{-1}{>}{}%
3730   },
3731 }
3732
3733 \def\forest@nodewalk@minmax#1#2#3#4{%
3734   % #1 = list of nodes
3735   % #2 = max index in list (start with 1)
3736   % #3 = min/max = ascending/descending = </>
3737   % #4 = how many results? 1 = {\forest@nodewalk@minmax@node}, all={\forest@nodewalk@minmax@nodes}, walk in
3738   \edef\forest@nodesort@sortkey{\forest@nodesort@by}%
3739   \edef\forest@nodewalk@minmax@N{\number\numexpr#2}%
3740   \edef\forest@nodewalk@minmax@n{%
3741     \edef\forest@nodewalk@minmax@list{#1}%
3742     \def\forest@nodewalk@minmax@nodes}%
3743   \def\forest@nodewalk@minmax@node}%
3744   \ifdefempty{\forest@nodewalk@minmax@list}{%
3745   }{%
3746     \safeloop
3747       \expandafter\forest@csvlist@getfirstrest@{\expandafter{\forest@nodewalk@minmax@list}\forest@nodewalk@min
3748       \ifnum\forest@nodewalk@minmax@cn=0 \else
3749         \ifdefempty{\forest@nodewalk@minmax@node}{%
3750           \edef\forest@nodewalk@minmax@node{\forest@nodewalk@minmax@cn}%
3751           \edef\forest@nodewalk@minmax@nodes{\forest@nodewalk@minmax@cn,%
3752           \edef\forest@nodewalk@minmax@n{\safeloopn}%
3753         }{%
3754           \csedef{forest@nodesort@1}{\forest@nodewalk@minmax@node}%
3755           \csedef{forest@nodesort@2}{\forest@nodewalk@minmax@cn}%
3756           \forest@nodesort@cmpnodes{2}{1}%
3757           \if=\forest@sort@cmp@result
3758             \edef\forest@nodewalk@minmax@node{\forest@nodewalk@minmax@cn}%
3759             \epreto\forest@nodewalk@minmax@nodes{\forest@nodewalk@minmax@cn,%
3760             \edef\forest@nodewalk@minmax@n{\safeloopn}%
3761           \else
3762             \if#3\forest@sort@cmp@result
3763               \edef\forest@nodewalk@minmax@node{\forest@nodewalk@minmax@cn}%
3764               \edef\forest@nodewalk@minmax@nodes{\forest@nodewalk@minmax@cn,%
3765               \edef\forest@nodewalk@minmax@n{\safeloopn}%
3766             \fi
3767           \fi
3768         }%
3769       \fi
3770       \ifdefempty{\forest@nodewalk@minmax@list}{\forest@tempfalse}{\forest@temptrue}%
3771       \ifnum\safeloopn=\forest@nodewalk@minmax@N\relax\forest@temptrue\fi
3772     \ifforest@temp
3773       \saferepeat
3774       \edef\forest@nodewalk@minmax@list{#4}%
3775       \ifdefempty\forest@nodewalk@minmax@list{%
3776         \forestset{nodewalk/jump back=\forest@nodewalk@minmax@n-1}% CHECK
3777       }{%
3778         \safeloop
3779           \expandafter\forest@csvlist@getfirstrest@{\expandafter{\forest@nodewalk@minmax@list}\forest@cn\forest@
3780           \forest@nodewalk@makestep
3781           \ifdefempty{\forest@nodewalk@minmax@list}{\forest@tempfalse}{\forest@temptrue}%
3782         \ifforest@temp
3783           \saferepeat
3784         }%
3785       }%
3786 }

```

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.

```

3787 \newtoks\forest@nodewalk@shortsteps@resolution
3788 \newif\ifforest@nodewalk@areshortsteps
3789 \pgfqkeys{/forest/nodewalk}{
3790   .unknown/.code={%
3791     \forest@nodewalk@areshortstepsfalse
3792     \pgfkeysifdefined{/forest/\pgfkeyscurrentname/.cmd}{%
3793       }{%
3794         \ifx\pgfkeyscurrentvalue\pgfkeysnovalue@text % no value, so possibly short steps
3795         \forest@nodewalk@shortsteps@resolution{}%
3796         \forest@nodewalk@areshortstepstrue
3797         \expandafter\forest@nodewalk@shortsteps\pgfkeyscurrentname=====,% "=" and "," cannot be short st
3798       \fi
3799     }%
3800     \ifforest@nodewalk@areshortsteps
3801     \@escapeif{\expandafter\pgfkeysalso\expandafter{\the\forest@nodewalk@shortsteps@resolution}}%
3802   \else
3803     \@escapeif{\pgfkeysalso{/forest/\pgfkeyscurrentname={#1}}}%
3804   \fi
3805 },
3806 }
3807 \def\forest@nodewalk@shortsteps{%
3808   \futurelet\forest@nodewalk@nexttoken\forest@nodewalk@shortsteps@
3809 }
3810 \def\forest@nodewalk@shortsteps@{%
3811   \ifx\forest@nodewalk@nexttoken=%
3812     \let\forest@nodewalk@nextop\forest@nodewalk@shortsteps@end
3813   \else
3814     \ifx\forest@nodewalk@nexttoken\bgroup
3815       \letcs\forest@nodewalk@nextop{forest@shortstep@group}%
3816     \else
3817       \let\forest@nodewalk@nextop\forest@nodewalk@shortsteps@@
3818     \fi
3819   \fi
3820   \forest@nodewalk@nextop
3821 }
3822 \def\forest@nodewalk@shortsteps@@#1{%
3823   \ifcsdef{forest@shortstep@#1}{%
3824     \csname forest@shortstep@#1\endcsname
3825   }{%
3826     \forest@nodewalk@areshortstepsfalse
3827     \forest@nodewalk@shortsteps@end
3828   }%
3829 }
3830 % in the following definitions:
3831 % #1 = short step
3832 % #2 = (long) step, or a style in /forest/nodewalk (taking n args)
3833 \csdef{forest@nodewalk@defshortstep@0@args}#1#2{%
3834   \csdef{forest@shortstep@#1}{%
3835     \apptotoks\forest@nodewalk@shortsteps@resolution{,#2}%
3836     \forest@nodewalk@shortsteps}}
3837 \csdef{forest@nodewalk@defshortstep@1@args}#1#2{%
3838   \csdef{forest@shortstep@#1}##1{%
3839     \edef\forest@marshal####1{#2}%
3840     \eapptotoks\forest@nodewalk@shortsteps@resolution{,\forest@marshal{##1}}%
3841     \forest@nodewalk@shortsteps}}
3842 \csdef{forest@nodewalk@defshortstep@2@args}#1#2{%
3843   \csdef{forest@shortstep@#1}##1##2{%
3844     \edef\forest@marshal####1####2{#2}%

```

```

3845 \eapptotoks\forest@nodewalk@shortsteps@resolution{,\forest@marshal{##1}{##2}}%
3846 \forest@nodewalk@shortsteps}}
3847 \csdef{forest@nodewalk@defshortstep@3@args}#1#2{%
3848 \csdef{forest@shortstep@#1}##1##2##3{%
3849 \edef\forest@marshal####1####2####3{#2}%
3850 \eapptotoks\forest@nodewalk@shortsteps@resolution{,\forest@marshal{##1}{##2}{##3}}%
3851 \forest@nodewalk@shortsteps}}
3852 \csdef{forest@nodewalk@defshortstep@4@args}#1#2{%
3853 \csdef{forest@shortstep@#1}##1##2##3##4{%
3854 \edef\forest@marshal####1####2####3####4{#2}%
3855 \eapptotoks\forest@nodewalk@shortsteps@resolution{,\forest@marshal{##1}{##2}{##3}{##4}}%
3856 \forest@nodewalk@shortsteps}}
3857 \csdef{forest@nodewalk@defshortstep@5@args}#1#2{%
3858 \csdef{forest@shortstep@#1}##1##2##3##4##5{%
3859 \edef\forest@marshal####1####2####3####4####5{#2}%
3860 \eapptotoks\forest@nodewalk@shortsteps@resolution{,\forest@marshal{##1}{##2}{##3}{##4}{##5}}%
3861 \forest@nodewalk@shortsteps}}
3862 \csdef{forest@nodewalk@defshortstep@6@args}#1#2{%
3863 \csdef{forest@shortstep@#1}##1##2##3##4##5##6{%
3864 \edef\forest@marshal####1####2####3####4####5####6{#2}%
3865 \eapptotoks\forest@nodewalk@shortsteps@resolution{,\forest@marshal{##1}{##2}{##3}{##4}{##5}{##6}}%
3866 \forest@nodewalk@shortsteps}}
3867 \csdef{forest@nodewalk@defshortstep@7@args}#1#2{%
3868 \csdef{forest@shortstep@#1}##1##2##3##4##5##6##7{%
3869 \edef\forest@marshal####1####2####3####4####5####6####7{#2}%
3870 \eapptotoks\forest@nodewalk@shortsteps@resolution{,\forest@marshal{##1}{##2}{##3}{##4}{##5}{##6}{##7}}%
3871 \forest@nodewalk@shortsteps}}
3872 \csdef{forest@nodewalk@defshortstep@8@args}#1#2{%
3873 \csdef{forest@shortstep@#1}##1##2##3##4##5##6##7##8{%
3874 \edef\forest@marshal####1####2####3####4####5####6####7####8{#2}%
3875 \eapptotoks\forest@nodewalk@shortsteps@resolution{,\forest@marshal{##1}{##2}{##3}{##4}{##5}{##6}{##7}{##8}}%
3876 \forest@nodewalk@shortsteps}}
3877 \csdef{forest@nodewalk@defshortstep@9@args}#1#2{%
3878 \csdef{forest@shortstep@#1}##1##2##3##4##5##6##7##8##9{%
3879 \edef\forest@marshal####1####2####3####4####5####6####7####8####9{#2}%
3880 \eapptotoks\forest@nodewalk@shortsteps@resolution{,\forest@marshal{##1}{##2}{##3}{##4}{##5}{##6}{##7}{##8}{##9}}%
3881 \forest@nodewalk@shortsteps}}
3882 \forestset{
3883 define short step/.code n args=3{% #1 = short step, #2 = n args, #3 = long step
3884 \csname forest@nodewalk@defshortstep@#2@args\endcsname{##1}{#3}%
3885 },
3886 }
3887 \def\forest@nodewalk@shortsteps@end#1,{
  Define short-form steps.
3888 \forestset{
3889 define short step={group}{1}{group={#1}}, % {braces} are special
3890 define short step={p}{0}{previous},
3891 define short step={n}{0}{next},
3892 define short step={u}{0}{parent},
3893 define short step={s}{0}{sibling},
3894 define short step={c}{0}{current},
3895 define short step={o}{0}{origin},
3896 define short step={r}{0}{root},
3897 define short step={R}{0}{root'},
3898 define short step={P}{0}{previous leaf},
3899 define short step={N}{0}{next leaf},
3900 define short step={F}{0}{first leaf},
3901 define short step={L}{0}{last leaf},
3902 define short step={>}{0}{next on tier},
3903 define short step={<}{0}{previous on tier},

```

```

3904 define short step={1}{0}{n=1},
3905 define short step={2}{0}{n=2},
3906 define short step={3}{0}{n=3},
3907 define short step={4}{0}{n=4},
3908 define short step={5}{0}{n=5},
3909 define short step={6}{0}{n=6},
3910 define short step={7}{0}{n=7},
3911 define short step={8}{0}{n=8},
3912 define short step={9}{0}{n=9},
3913 define short step={l}{0}{last},
3914 define short step={b}{0}{back},
3915 define short step={f}{0}{forward},
3916 define short step={v}{0}{last valid},
3917 define short step={*}{2}{repeat={#1}{#2}},
3918 for 1/.style={for nodewalk={n=1}{#1}},
3919 for 2/.style={for nodewalk={n=2}{#1}},
3920 for 3/.style={for nodewalk={n=3}{#1}},
3921 for 4/.style={for nodewalk={n=4}{#1}},
3922 for 5/.style={for nodewalk={n=5}{#1}},
3923 for 6/.style={for nodewalk={n=6}{#1}},
3924 for 7/.style={for nodewalk={n=7}{#1}},
3925 for 8/.style={for nodewalk={n=8}{#1}},
3926 for 9/.style={for nodewalk={n=9}{#1}},
3927 for -1/.style={for nodewalk={n'=1}{#1}},
3928 for -2/.style={for nodewalk={n'=2}{#1}},
3929 for -3/.style={for nodewalk={n'=3}{#1}},
3930 for -4/.style={for nodewalk={n'=4}{#1}},
3931 for -5/.style={for nodewalk={n'=5}{#1}},
3932 for -6/.style={for nodewalk={n'=6}{#1}},
3933 for -7/.style={for nodewalk={n'=7}{#1}},
3934 for -8/.style={for nodewalk={n'=8}{#1}},
3935 for -9/.style={for nodewalk={n'=9}{#1}},
3936 }

```

Define multiple-step walks.

```

3937 \forestset{
3938   define long step={tree}{\forest@node@foreach{\forest@nodewalk@makestep}},
3939   define long step={tree reversed}{\forest@node@foreach@reversed{\forest@nodewalk@makestep}},
3940   define long step={tree children-first}{\forest@node@foreach@childrenfirst{\forest@nodewalk@makestep}},
3941   define long step={tree children-first reversed}{\forest@node@foreach@childrenfirst@reversed{\forest@nodewalk@makestep}},
3942   define long step={tree breadth-first}{\forest@node@foreach@breadthfirst{-1}{\forest@nodewalk@makestep}},
3943   define long step={tree breadth-first reversed}{\forest@node@foreach@breadthfirst@reversed{-1}{\forest@nodewalk@makestep}},
3944   define long step={descendants}{\forest@node@foreach@descendant{\forest@nodewalk@makestep}},
3945   define long step={descendants reversed}{\forest@node@foreach@descendant@reversed{\forest@nodewalk@makestep}},
3946   define long step={descendants children-first}{\forest@node@foreach@descendant@childrenfirst{\forest@nodewalk@makestep}},
3947   define long step={descendants children-first reversed}{\forest@node@foreach@descendant@childrenfirst@reversed{\forest@nodewalk@makestep}},
3948   define long step={descendants breadth-first}{\forest@node@foreach@breadthfirst{0}{\forest@nodewalk@makestep}},
3949   define long step={descendants breadth-first reversed}{\forest@node@foreach@breadthfirst@reversed{0}{\forest@nodewalk@makestep}},
3950   define long step={level}{n args=1}{%
3951     \pgfmathtruncatemacro\forest@temp{#1}%
3952     \edef\forest@marshal{%
3953       \noexpand\forest@node@foreach@breadthfirst
3954       {\forest@temp}%
3955       {\noexpand\ifnum\noexpand\forest@level=\forest@temp\relax\noexpand\forest@nodewalk@makestep\noexpand\forest@temp\relax}%
3956     }\forest@marshal
3957   },
3958   define long step={level>}{n args=1}{%
3959     \pgfmathtruncatemacro\forest@temp{#1}%
3960     \edef\forest@marshal{%
3961       \noexpand\forest@node@foreach@breadthfirst
3962       {-1}%

```

```

3963     {\noexpand\ifnum\noexpand\forestove{level}<\forest@temp\relax\noexpand\else\noexpand\forest@nodewalk@
3964 } \forest@marshal
3965 },
3966 define long step={level<}{n args=1}{%
3967   \pgfmathtruncatemacro\forest@temp{(#1)-1}%
3968   \edef\forest@marshal{%
3969     \noexpand\forest@node@foreach@breadthfirst
3970     {\forest@temp}%
3971     {\noexpand\forest@nodewalk@makestep}%
3972   } \forest@marshal
3973 },
3974 define long step={level reversed}{n args=1}{%
3975   \pgfmathtruncatemacro\forest@temp{#1}%
3976   \edef\forest@marshal{%
3977     \noexpand\forest@node@foreach@breadthfirst@reversed
3978     {\forest@temp}%
3979     {\noexpand\ifnum\noexpand\forestove{level}=\forest@temp\relax\noexpand\forest@nodewalk@makestep\noexp
3980   } \forest@marshal
3981 },
3982 define long step={level reversed>}{n args=1}{%
3983   \pgfmathtruncatemacro\forest@temp{#1}%
3984   \edef\forest@marshal{%
3985     \noexpand\forest@node@foreach@breadthfirst@reversed
3986     {-1}%
3987     {\noexpand\ifnum\noexpand\forestove{level}<\forest@temp\relax\noexpand\else\noexpand\forest@nodewalk@
3988   } \forest@marshal
3989 },
3990 define long step={level reversed<}{n args=1}{%
3991   \pgfmathtruncatemacro\forest@temp{(#1)-1}%
3992   \edef\forest@marshal{%
3993     \noexpand\forest@node@foreach@breadthfirst@reversed
3994     {\forest@temp}%
3995     {\noexpand\forest@nodewalk@makestep}%
3996   } \forest@marshal
3997 },
3998 %
3999 define long step={relative level}{n args=1}{%
4000   \pgfmathtruncatemacro\forest@temp{(#1)+\forestove{level}}%
4001   \edef\forest@marshal{%
4002     \noexpand\forest@node@foreach@breadthfirst
4003     {\forest@temp}%
4004     {\noexpand\ifnum\noexpand\forestove{level}=\forest@temp\relax\noexpand\forest@nodewalk@makestep\noexp
4005   } \forest@marshal
4006 },
4007 define long step={relative level>}{n args=1}{%
4008   \pgfmathtruncatemacro\forest@temp{(#1)+\forestove{level}}%
4009   \edef\forest@marshal{%
4010     \noexpand\forest@node@foreach@breadthfirst
4011     {-1}%
4012     {\noexpand\ifnum\noexpand\forestove{level}<\forest@temp\relax\noexpand\else\noexpand\forest@nodewalk@
4013   } \forest@marshal
4014 },
4015 define long step={relative level<}{n args=1}{%
4016   \pgfmathtruncatemacro\forest@temp{(#1)+\forestove{level}-1}%
4017   \edef\forest@marshal{%
4018     \noexpand\forest@node@foreach@breadthfirst
4019     {\forest@temp}%
4020     {\noexpand\forest@nodewalk@makestep}%
4021   } \forest@marshal
4022 },
4023 define long step={relative level reversed}{n args=1}{%

```



```

4024 \pgfmathtruncatemacro\forest@temp{(#1)+\forest@ve{level}}%
4025 \edef\forest@marshal{%
4026 \noexpand\forest@node@foreach@breadthfirst@reversed
4027 {\forest@temp}%
4028 {\noexpand\ifnum\noexpand\forest@ve{level}=\forest@temp\relax\noexpand\forest@nodewalk@makestep\noexp
4029 }\forest@marshal
4030 },
4031 define long step={relative level reversed>}{n args=1}{%
4032 \pgfmathtruncatemacro\forest@temp{(#1)+\forest@ve{level}}%
4033 \edef\forest@marshal{%
4034 \noexpand\forest@node@foreach@breadthfirst@reversed
4035 {-1}%
4036 {\noexpand\ifnum\noexpand\forest@ve{level}<\forest@temp\relax\noexpand\else\noexpand\forest@nodewalk@
4037 }\forest@marshal
4038 },
4039 define long step={relative level reversed<}{n args=1}{%
4040 \pgfmathtruncatemacro\forest@temp{(#1)+\forest@ve{level}-1}%
4041 \edef\forest@marshal{%
4042 \noexpand\forest@node@foreach@breadthfirst@reversed
4043 {\forest@temp}%
4044 {\noexpand\forest@nodewalk@makestep}%
4045 }\forest@marshal
4046 },
4047 define long step={children}{\forest@node@foreachchild{\forest@nodewalk@makestep}},
4048 define long step={children reversed}{\forest@node@foreachchild@reversed{\forest@nodewalk@makestep}},
4049 define long step={current and following siblings}{\forest@node@@forselfandfollowingsiblings{\forest@nodew
4050 define long step={following siblings}{style}{if nodewalk valid={next}{fake=next,current and following sibli
4051 define long step={current and preceding siblings}{\forest@node@@forselfandprecedingsiblings{\forest@nodew
4052 define long step={preceding siblings}{style}{if nodewalk valid={previous}{fake=previous,current and precedi
4053 define long step={current and following siblings reversed}{\forest@node@@forselfandfollowingsiblings@reve
4054 define long step={following siblings reversed}{style}{fake=next,current and following siblings reversed},
4055 define long step={current and preceding siblings reversed}{\forest@node@@forselfandprecedingsiblings@reve
4056 define long step={preceding siblings reversed}{style}{fake=previous,current and preceding siblings reversed
4057 define long step={siblings}{style}{for nodewalk'={preceding siblings},following siblings},
4058 define long step={siblings reversed}{style}{for nodewalk'={following siblings reversed},preceding siblings
4059 define long step={current and siblings}{style}{for nodewalk'={preceding siblings},current and following sib
4060 define long step={current and siblings reversed}{style}{for nodewalk'={current and following siblings rever
4061 define long step={ancestors}{style}{while={}{parent},last valid},
4062 define long step={current and ancestors}{style}{current,ancestors},
4063 define long step={following nodes}{style}{while={}{next node},last valid},
4064 define long step={preceding nodes}{style}{while={}{previous node},last valid},
4065 define long step={current and following nodes}{style}{current,following nodes},
4066 define long step={current and preceding nodes}{style}{current,preceding nodes},
4067 }
4068 \let\forest@nodewalkstephandler@styletrueorfalse\forest@nodewalkstephandler@styletrue

```

6.6 Dynamic tree

```

4069 \def\forest@last@node{0}
4070 \def\forest@nodehandleby@name@nodewalk@or@bracket#1{%
4071 \ifx\pgfkeysnovalue#1%
4072 \edef\forest@last@node{\forest@node@Nametoid{forest@last@node}}%
4073 \else
4074 \forest@nodehandleby@nnb@checkfirst#1\forest@END
4075 \fi
4076 }
4077 \def\forest@nodehandleby@nnb@checkfirst#1#2\forest@END{%
4078 \ifx[#1]
4079 \forest@create@node{#1#2}%
4080 \else

```

```

4081 \forest@forthis{%
4082 \forest@nameandgo{#1#2}%
4083 \ifnum\forest@cn=0
4084 \PackageError{forest}{Cannot use a dynamic key on the invalid node}{}%
4085 \fi
4086 \let\forest@last@node\forest@cn
4087 }%
4088 \fi
4089 }
4090 \def\forest@create@node#1{% #1=bracket representation
4091 \bracketParse{\forest@create@collectafterthought}%
4092 \forest@last@node=#1\forest@end@create@node
4093 }
4094 \def\forest@create@collectafterthought#1\forest@end@create@node{%
4095 \forest@node@Foreach{\forest@last@node}{%
4096 \forest@to{delay}{given options}%
4097 \forest@set{given options}{}%
4098 }%
4099 \forest@eappto{\forest@last@node}{delay}{,\unexpanded{#1}}%
4100 \forest@set{\forest@last@node}{given options}{delay={}}%
4101 }
4102 \def\forest@create@node@and@process@given@options#1{% #1=bracket representation
4103 \bracketParse{\forest@createandprocess@collectafterthought}%
4104 \forest@last@node=#1\forest@end@create@node
4105 }
4106 \def\forest@createandprocess@collectafterthought#1\forest@end@create@node{%
4107 \forest@node@Compute@numeric@ts@info{\forest@last@node}%
4108 \forest@saveandrestoremacro\forest@root{%
4109 \let\forest@root\forest@last@node
4110 \forest@set{process keylist=given options}%
4111 }%
4112 }
4113 \def\forest@saveandrestoremacro#1#2{% #1 = the (zero-arg) macro to save before and restore after processing c
4114 \edef\forest@marshal{%
4115 \unexpanded{#2}%
4116 \noexpand\def\noexpand#1{\expandonce{#1}}%
4117 }\forest@marshal
4118 }
4119 \def\forest@saveandrestoreifcs#1#2{% #1 = the if cs to save before and restore after processing code in #2
4120 \edef\forest@marshal{%
4121 \unexpanded{#2}%
4122 \ifbool{#1}{\noexpand\setbool{#1}{true}}{\noexpand\setbool{#1}{false}}
4123 }\forest@marshal
4124 }
4125 \def\forest@saveandrestoretoks#1#2{% #1 = the toks to save before and restore after processing code in #2
4126 \edef\forest@marshal{%
4127 \unexpanded{#2}%
4128 \noexpand#1{\the#1}%
4129 }\forest@marshal
4130 }
4131 \def\forest@saveandrestoreregister#1#2{% #1 = the register to save before and restore after processing code i
4132 \edef\forest@marshal{%
4133 \unexpanded{#2}%
4134 \noexpand\forest@rset{#1}{\forestregister{#1}}%
4135 }\forest@marshal
4136 }
4137 \def\forest@remove@node#1{%
4138 \forest@node@Remove{#1}%
4139 }
4140 \def\forest@append@node#1#2{%
4141 \forest@node@Remove{#2}%

```

```

4142 \forest@node@Append{#1}{#2}%
4143 }
4144 \def\forest@prepend@node#1#2{%
4145 \forest@node@Remove{#2}%
4146 \forest@node@Prepend{#1}{#2}%
4147 }
4148 \def\forest@insertafter@node#1#2{%
4149 \forest@node@Remove{#2}%
4150 \forest@node@Insertafter{\forestOve{#1}{@parent}}{#2}{#1}%
4151 }
4152 \def\forest@insertbefore@node#1#2{%
4153 \forest@node@Remove{#2}%
4154 \forest@node@Insertbefore{\forestOve{#1}{@parent}}{#2}{#1}%
4155 }
4156 \def\forest@set@root#1#2{%
4157 \def\forest@root{#2}%
4158 }
4159 \def\forest@appto@do@ynamics#1#2{%
4160 \forest@nodehandleby@name@nodewalk@or@bracket{#2}%
4161 \ifcase\forest@ynamics@copyhow\relax\or
4162 \forest@tree@copy{\forest@last@node}\forest@last@node
4163 \or
4164 \forest@node@copy{\forest@last@node}\forest@last@node
4165 \fi
4166 \forest@node@Ifnamedefined{\forest@last@node}{%
4167 \forest@preto{\forest@last@node}{delay}
4168 {for id={\forest@node@Nametoid{\forest@last@node}}{alias=\forest@last@node}},}%
4169 }{%
4170 \edef\forest@marshal{%
4171 \noexpand\apptotoks\noexpand\forest@do@ynamics{%
4172 \noexpand#1{\forest@cn}{\forest@last@node}}%
4173 }\forest@marshal
4174 }
4175 \forestset{%
4176 create/.code={%
4177 \forest@create@node{#1}%
4178 \forest@for@node{\forest@last@node}{\forest@node@setalias{\forest@last@node}}%
4179 },
4180 create'/.code={%
4181 \forest@create@node@and@process@given@options{#1}%
4182 \forest@for@node{\forest@last@node}{\forest@node@setalias{\forest@last@node}}%
4183 },
4184 append/.code={\def\forest@ynamics@copyhow{0}\forest@appto@do@ynamics\forest@append@node{#1}},
4185 prepend/.code={\def\forest@ynamics@copyhow{0}\forest@appto@do@ynamics\forest@prepend@node{#1}},
4186 insert after/.code={\def\forest@ynamics@copyhow{0}\forest@appto@do@ynamics\forest@insertafter@node{#1}},
4187 insert before/.code={\def\forest@ynamics@copyhow{0}\forest@appto@do@ynamics\forest@insertbefore@node{#1}},
4188 append'/.code={\def\forest@ynamics@copyhow{1}\forest@appto@do@ynamics\forest@append@node{#1}},
4189 prepend'/.code={\def\forest@ynamics@copyhow{1}\forest@appto@do@ynamics\forest@prepend@node{#1}},
4190 insert after'/.code={\def\forest@ynamics@copyhow{1}\forest@appto@do@ynamics\forest@insertafter@node{#1}},
4191 insert before'/.code={\def\forest@ynamics@copyhow{1}\forest@appto@do@ynamics\forest@insertbefore@node{#1}},
4192 append''/.code={\def\forest@ynamics@copyhow{2}\forest@appto@do@ynamics\forest@append@node{#1}},
4193 prepend''/.code={\def\forest@ynamics@copyhow{2}\forest@appto@do@ynamics\forest@prepend@node{#1}},
4194 insert after''/.code={\def\forest@ynamics@copyhow{2}\forest@appto@do@ynamics\forest@insertafter@node{#1}},
4195 insert before''/.code={\def\forest@ynamics@copyhow{2}\forest@appto@do@ynamics\forest@insertbefore@node{#1}},
4196 remove/.code={%
4197 \pgfkeysalso{alias=\forest@last@node}%
4198 \expandafter\apptotoks\expandafter\forest@do@ynamics\expandafter{%
4199 \expandafter\forest@remove@node\expandafter{\forest@cn}}%
4200 },
4201 set root/.code={%
4202 \def\forest@ynamics@copyhow{0}%

```

```

4203 \forest@appto@do@ynamics\forest@set@root{#1}%
4204 },
4205 replace by/.code={\forest@replaceby@code{#1}{insert after}},
4206 replace by'/.code={\forest@replaceby@code{#1}{insert after'}},
4207 replace by''/.code={\forest@replaceby@code{#1}{insert after''}},
4208 sort/.code={%
4209 \eapptotoks\forest@do@ynamics{%
4210 \noexpand\forest@nodesort
4211 \noexpand\forest@sort@ascending
4212 {\forest@cn}%
4213 {\expandonce{\forest@nodesort@by}}}%
4214 }%
4215 },
4216 sort'/.code={%
4217 \eapptotoks\forest@do@ynamics{%
4218 \noexpand\forest@nodesort
4219 \noexpand\forest@sort@descending
4220 {\forest@cn}%
4221 {\expandonce{\forest@nodesort@by}}}%
4222 }%
4223 },
4224 sort by/.store in=\forest@nodesort@by,
4225 }
4226 \def\forest@replaceby@code#1#2{%#1=node spec,#2=insert after['']['']}
4227 \ifnum\forestove{@parent}=0
4228 \pgfkeysalso{set root={#1}}%
4229 \else
4230 \pgfkeysalso{alias=forest@last@node,#2={#1}}%
4231 \eapptotoks\forest@do@ynamics{%
4232 \noexpand\ifnum\noexpand\forestOve{\forest@cn}{@parent}=\forestove{@parent}
4233 \noexpand\forest@remove@node{\forest@cn}%
4234 \noexpand\fi
4235 }%
4236 \fi
4237 }
4238 \def\forest@nodesort#1#2#3{% #1 = direction, #2 = parent node, #3 = sort key
4239 \def\forest@nodesort@sortkey{#3}%
4240 \forest@for@node{#2}{\forest@nodesort@#1}%
4241 }
4242 \def\forest@nodesort@#1{%
4243 % prepare the array of child ids
4244 \c@pgf@counta=0
4245 \forest@toget{@first}\forest@nodesort@id
4246 \forest@loop
4247 \ifnum\forest@nodesort@id>0
4248 \advance\c@pgf@counta 1
4249 \csedef{\forest@nodesort@the\c@pgf@counta}{\forest@nodesort@id}%
4250 \forest@get{\forest@nodesort@id}{@next}\forest@nodesort@id
4251 \forest@repeat
4252 % sort
4253 \forest@toget{n children}\forest@nodesort@n
4254 \forest@sort\forest@nodesort@cmpnodes\forest@nodesort@let#1{1}{\forest@nodesort@n}%
4255 % remove all children
4256 \forest@toget{@first}\forest@nodesort@id
4257 \forest@loop
4258 \ifnum\forest@nodesort@id>0
4259 \forest@node@Remove{\forest@nodesort@id}%
4260 \forest@toget{@first}\forest@nodesort@id
4261 \forest@repeat
4262 % insert the children in new order
4263 \c@pgf@counta=0

```

```

4264 \forest@loop
4265 \ifnum\c@pgf@counta<\forest@nodesort@n\relax
4266 \advance\c@pgf@counta 1
4267 \edef\temp{\csname forest@nodesort@the\c@pgf@counta\endcsname}%
4268 \forest@node@append{\csname forest@nodesort@the\c@pgf@counta\endcsname}%
4269 \forest@repeat
4270 }
4271 \def\forest@nodesort@cmpnodes#1#2{%
4272 \global\let\forest@nodesort@cmpresult\forest@sort@cmp@eq
4273 \foreach \forest@temp@pgfmth in \forest@nodesort@sortkey {%
4274 \forest@fornode{\csname forest@nodesort@#1\endcsname}{%
4275 \pgfmthparse{\forest@temp@pgfmth}\global\let\forest@global@tempa\pgfmthresult}%
4276 \forest@fornode{\csname forest@nodesort@#2\endcsname}{%
4277 \pgfmthparse{\forest@temp@pgfmth}\global\let\forest@global@tempb\pgfmthresult}%
4278 \ifdim\forest@global@tempa pt<\forest@global@tempb pt
4279 \global\let\forest@nodesort@cmpresult\forest@sort@cmp@lt
4280 \breakforeach
4281 \else
4282 \ifdim\forest@global@tempa pt>\forest@global@tempb pt
4283 \global\let\forest@nodesort@cmpresult\forest@sort@cmp@gt
4284 \breakforeach
4285 \fi
4286 \fi
4287 }%
4288 \forest@nodesort@cmpresult
4289 }
4290 \def\forest@nodesort@let#1#2{%
4291 \csletcs{forest@nodesort@#1}{forest@nodesort@#2}%
4292 }

```

7 Stages

```

4293 \def\forest@root{0}
4294 %%% begin listing region: stages
4295 \forestset{
4296 stages/.style={
4297 for root'={
4298 process keylist register=default preamble,
4299 process keylist register=preamble
4300 },
4301 process keylist=given options,
4302 process keylist=before typesetting nodes,
4303 typeset nodes stage,
4304 process keylist=before packing,
4305 pack stage,
4306 process keylist=before computing xy,
4307 compute xy stage,
4308 process keylist=before drawing tree,
4309 draw tree stage
4310 },
4311 typeset nodes stage/.style={for root'=typeset nodes},
4312 pack stage/.style={for root'=pack},
4313 compute xy stage/.style={for root'=compute xy},
4314 draw tree stage/.style={for root'=draw tree},
4315 }
4316 %%% end listing region: stages
4317 \forestset{
4318 process keylist/.code={%
4319 \forest@process@hook@keylist{#1}{#1 processing order/.try,processing order/.lastretry}},
4320 process keylist'/.code 2 args={\forest@process@hook@keylist@nodynamics{#1}{#2}},
4321 process keylist''/.code 2 args={\forest@process@hook@keylist@{#1}{#2}},

```

```

4322 process keylist register/.code={\forest@process@keylist@register{#1}},
4323 process delayed/.code={%
4324   \forest@havedelayedoptions{@delay}{#1}%
4325   \ifforest@havedelayedoptions
4326     \forest@process@hook@keylist@nodynamics{@delay}{#1}%
4327   \fi
4328 },
4329 do dynamics/.code={%
4330   \the\forest@do@dynamics
4331   \forest@do@dynamics{}%
4332   \forest@node@Compute@numeric@ts@info{\forest@root}%
4333 },
4334 declare keylist={given options}{},
4335 declare keylist={before typesetting nodes}{},
4336 declare keylist={before packing}{},
4337 declare keylist={before packing node}{},
4338 declare keylist={after packing node}{},
4339 declare keylist={before computing xy}{},
4340 declare keylist={before drawing tree}{},
4341 declare keylist={delay}{},
4342 delay n/.style 2 args={if={#1==0}{#2}{delay@n={#1}{#2}}},
4343 delay@n/.style 2 args={
4344   if={#1==1}{delay={#2}}{delay={delay@n/.wrap pgfmath arg={{##1}{#2}}{#1-1}}}
4345 },
4346 if have delayed/.style 2 args={if have delayed'={processing order}{#1}{#2}},
4347 if have delayed'/.code n args=3{%
4348   \forest@havedelayedoptionsfalse
4349   \forest@forthis{%
4350     \forest@nodewalk{#1}{%
4351       TeX={%
4352         \forest@get{delay}\forest@temp@delayed
4353         \ifdefempty\forest@temp@delayed{}{\forest@havedelayedoptionstrue}%
4354       }%
4355     }%
4356   }%
4357   \ifforest@havedelayedoptions\pgfkeysalso{#2}\else\pgfkeysalso{#3}\fi
4358 },
4359 typeset nodes/.code={%
4360   \forest@drawtree@preservenodeboxes@false
4361   \forest@nodewalk
4362     {typeset nodes processing order/.try,processing order/.lastretry}%
4363     {TeX={\forest@node@typeset}}%
4364 },
4365 typeset nodes'/.code={%
4366   \forest@drawtree@preservenodeboxes@true
4367   \forest@nodewalk
4368     {typeset nodes processing order/.try,processing order/.lastretry}%
4369     {TeX={\forest@node@typeset}}%
4370 },
4371 typeset node/.code={%
4372   \forest@drawtree@preservenodeboxes@false
4373   \forest@node@typeset
4374 },
4375 pack/.code={\forest@pack},
4376 pack'/.code={\forest@pack@onlythisnode},
4377 compute xy/.code={\forest@node@computeabsolute positions},
4378 draw tree box/.store in=\forest@drawtreebox,
4379 draw tree box,
4380 draw tree/.code={%
4381   \forest@drawtree@preservenodeboxes@false
4382   \forest@node@drawtree

```

```

4383 },
4384 draw tree'/.code={%
4385   \forest@drawtree@preservenodeboxes@true
4386   \forest@node@drawtree
4387 },
4388 %%% begin listing region: draw_tree_method
4389 draw tree method/.style={
4390   for nodewalk={
4391     draw tree nodes processing order/.try,
4392     draw tree processing order/.retry,
4393     processing order/.lastretry
4394   }{draw tree node},
4395   for nodewalk={
4396     draw tree edges processing order/.try,
4397     draw tree processing order/.retry,
4398     processing order/.lastretry
4399   }{draw tree edge},
4400   for nodewalk={
4401     draw tree tikz processing order/.try,
4402     draw tree processing order/.retry,
4403     processing order/.lastretry
4404   }{draw tree tikz}
4405 },
4406 %%% end listing region: draw_tree_method
4407 draw tree node/.code={\forest@draw@node},
4408 draw tree edge/.code={\forest@draw@edge},
4409 draw tree tikz/.code={\forest@draw@tikz},
4410 for nodewalk={processing order/.style={tree}}{ },
4411 %given options processing order/.style={processing order},
4412 %before typesetting nodes processing order/.style={processing order},
4413 %before packing processing order/.style={processing order},
4414 %before computing xy processing order/.style={processing order},
4415 %before drawing tree processing order/.style={processing order},
4416 }
4417 \newtoks\forest@do@ dynamics
4418 \newif\ifforest@havedelayedoptions
4419 \def\forest@process@hook@keylist#1#2{% ,#1=keylist,#2=processing order nodewalk
4420   \safeloop
4421     \forest@for node{\forest@root}{\forest@process@hook@keylist@{#1}{#2}}%
4422     \expandafter\ifstreampy\expandafter{\the\forest@do@ dynamics}{ }{%
4423       \the\forest@do@ dynamics
4424       \forest@do@ dynamics={ }%
4425       \forest@node@Compute@numeric@ts@info{\forest@root}%
4426     }%
4427   \forest@for node{\forest@root}{\forest@havedelayedoptions{#1}{#2}}%
4428   \ifforest@havedelayedoptions
4429   \saferepeat
4430 }
4431 \def\forest@process@hook@keylist@nodynamics#1#2{% ,#1=keylist,#2=processing order nodewalk
4432   % note: this macro works on (nodewalk starting at) the current node
4433   \safeloop
4434     \forest@for this{\forest@process@hook@keylist@{#1}{#2}}%
4435     \forest@havedelayedoptions{#1}{#2}%
4436     \ifforest@havedelayedoptions
4437     \saferepeat
4438 }
4439 \def\forest@process@hook@keylist@#1#2{% ,#1=keylist,#2=processing order nodewalk
4440   \forest@nodewalk{#2}{%
4441     TeX={%
4442       \forest@toget{#1}\forest@temp@keys
4443       \ifdefvoid\forest@temp@keys}{%

```

```

4444     \forestset{#1}{}%
4445     \expandafter\forestset\expandafter{\forest@temp@keys}%
4446 }%
4447 }%
4448 }%
4449 }
4450 \def\forest@process@keylist@register#1{%
4451   \edef\forest@marshal{%
4452     \noexpand\forestset{\forestregister{#1}}%
4453   }\forest@marshal
4454 }

```

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

```

4455 \def\forest@havedelayedoptions#1#2{%#1 = keylist, #2=nodewalk
4456   \forest@havedelayedoptionsfalse
4457   \forest@forthis{%
4458     \forest@nodewalk{#2}{%
4459       TeX={%
4460         \foresttoget{delay}\forest@temp@delayed
4461         \ifdefempty\forest@temp@delayed{}\forest@havedelayedoptionstrue}%
4462         \forestolet{#1}\forest@temp@delayed
4463         \forestoset{delay}{}%
4464       }%
4465     }%
4466   }%
4467 }

```

7.1 Typesetting nodes

```

4468 \def\forest@node@typeset{%
4469   \let\forest@next\forest@node@typeset@
4470   \forestoifdefined{@box}{%
4471     \foresttoget{@box}\forest@temp
4472     \ifdefempty\forest@temp{%
4473       \locbox\forest@temp@box
4474       \forestolet{@box}\forest@temp@box
4475     }{%
4476       \ifforest@drawtree@preservenodeboxes@
4477       \let\forest@next\relax
4478       \fi
4479     }%
4480   }{%
4481     \locbox\forest@temp@box
4482     \forestolet{@box}\forest@temp@box
4483   }%
4484   \def\forest@node@typeset@restore{}%
4485   \ifdefined\ifsa@tikz\forest@standalone@hack\fi
4486   \forest@next
4487   \forest@node@typeset@restore
4488 }
4489 \def\forest@standalone@hack{%
4490   \ifsa@tikz
4491     \let\forest@standalone@tikzpicture\tikzpicture
4492     \let\forest@standalone@endtikzpicture\endtikzpicture
4493     \let\tikzpicture\sa@orig@tikzpicture
4494     \let\endtikzpicture\sa@orig@endtikzpicture
4495     \def\forest@node@typeset@restore{%
4496       \let\tikzpicture\forest@standalone@tikzpicture
4497       \let\endtikzpicture\forest@standalone@endtikzpicture
4498     }%
4499     \fi
4500 }

```



```

4501 \newbox\forest@box
4502 \def\forest@pgf@notyetpositioned{not yet positionedPGFINTERNAL}
4503 \def\forest@node@typeset@{%
4504   \forestanchortotikzanchor{anchor}\forest@temp
4505   \edef\forest@marshal{%
4506     \noexpand\forestolet{anchor}\noexpand\forest@temp
4507     \noexpand\forest@node@typeset@@
4508     \noexpand\forestoset{anchor}{\forestov{anchor}}}%
4509   }\forest@marshal
4510 }
4511 \def\forest@node@typeset@@{%
4512   \forestoget{name}\forest@nodename
4513   \edef\forest@temp@nodeformat{\forestove{node format}}%
4514   \gdef\forest@smuggle{}%
4515   \setbox0=\hbox{%
4516     \begin{tikzpicture}[%
4517       /forest/copy command key={/tikz/anchor}/{/tikz/forest@orig@anchor},
4518       anchor/.style={%
4519         /forest/TeX={\forestanchortotikzanchor{##1}\forest@temp@anchor},
4520         forest@orig@anchor/.expand once=\forest@temp@anchor
4521       }]
4522     \pgfpositionnodelater{\forest@positionnodelater@save}%
4523     \forest@temp@nodeformat
4524     \pgfinterruptpath
4525     \pgfpointanchor{\forest@pgf@notyetpositioned\forest@nodename}{forestcomputenodeboundary}%
4526     \endpgfinterruptpath
4527     \end{tikzpicture}%
4528   }%
4529   \setbox\forestove{@box}=\box\forest@box % smuggle the box
4530   \forestolet{@boundary}\forest@global@boundary
4531   \forest@smuggle % ... and the rest
4532 }
4533
4534
4535 \forestset{
4536   declare readonly dimen={min x},
4537   declare readonly dimen={min y},
4538   declare readonly dimen={max x},
4539   declare readonly dimen={max y},
4540 }
4541 \def\forest@patch@enormouscoordinateboxbounds@plus#1{%
4542   \expandafter\ifstrequal\expandafter{#1}{16000.0pt}{\def#1{0.0pt}}{}%
4543 }
4544 \def\forest@patch@enormouscoordinateboxbounds@minus#1{%
4545   \expandafter\ifstrequal\expandafter{#1}{-16000.0pt}{\def#1{0.0pt}}{}%
4546 }
4547 \def\forest@positionnodelater@save{%
4548   \global\setbox\forest@box=\box\pgfpositionnodelaterbox
4549   \xappto\forest@smuggle{\noexpand\forestoset{later@name}{\pgfpositionnodelatername}}%
4550   % a bug in pgf? ---well, here's a patch
4551   \forest@patch@enormouscoordinateboxbounds@plus\pgfpositionnodelaterminx
4552   \forest@patch@enormouscoordinateboxbounds@plus\pgfpositionnodelaterminy
4553   \forest@patch@enormouscoordinateboxbounds@minus\pgfpositionnodelatermaxx
4554   \forest@patch@enormouscoordinateboxbounds@minus\pgfpositionnodelatermaxy
4555   % end of patch
4556   \xappto\forest@smuggle{\noexpand\forestoset{min x}{\pgfpositionnodelaterminx}}%
4557   \xappto\forest@smuggle{\noexpand\forestoset{min y}{\pgfpositionnodelaterminy}}%
4558   \xappto\forest@smuggle{\noexpand\forestoset{max x}{\pgfpositionnodelatermaxx}}%
4559   \xappto\forest@smuggle{\noexpand\forestoset{max y}{\pgfpositionnodelatermaxy}}%
4560 }
4561 \def\forest@node@forest@positionnodelater@restore{%

```

```

4562 \ifforest@drawtree@preservenodeboxes@
4563   \let\forest@boxorcopy\copy
4564 \else
4565   \let\forest@boxorcopy\box
4566 \fi
4567 \forestoget{@box}\forest@temp
4568 \setbox\pgfpositionnodelaterbox=\forest@boxorcopy\forest@temp
4569 \edef\pgfpositionnodelatername{\forestove{later@name}}%
4570 \edef\pgfpositionnodelaterminx{\forestove{min x}}%
4571 \edef\pgfpositionnodelaterminy{\forestove{min y}}%
4572 \edef\pgfpositionnodelatermaxx{\forestove{max x}}%
4573 \edef\pgfpositionnodelatermaxy{\forestove{max y}}%
4574 \ifforest@drawtree@preservenodeboxes@
4575 \else
4576   \forestoset{@box}{}%
4577 \fi
4578 }

```

7.2 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.

```

4579 \def\forest@pack{%
4580   \forest@pack@computetiers
4581   \forest@pack@computegrowthuniformity
4582   \forest@@@pack
4583 }
4584 \def\forest@@@pack{%
4585   \ifnum\forestove{n children}>0
4586     \ifnum\forestove{uniform growth}>0
4587       \forest@pack@level@uniform
4588       \forest@pack@aligntiers@ofsubtree
4589       \forest@pack@sibling@uniform@recursive
4590     \else
4591       \forest@node@foreachchild{\forest@@@pack}%
4592       \forest@process@hook@keylist@nodynamics{before packing node}{current}%
4593       \forest@pack@level@nonuniform
4594       \forest@pack@aligntiers
4595       \forest@pack@sibling@uniform@applyreversed
4596     \fi
4597   \fi
4598   \forestoget{after packing node}\forest@temp@keys
4599   \forest@process@hook@keylist@nodynamics{after packing node}{current}%
4600 }
4601 % \forestset{recalculate tree boundary/.code={\forest@node@recalculate@edges}}
4602 % \def\forest@node@recalculate@edges{%
4603 %   \edef\forest@marshal{%
4604 %     \noexpand\forest@forthis{\noexpand\forest@node@getedges{\forestove{grow}}}%
4605 %   }\forest@marshal
4606 % }
4607 \def\forest@pack@onlythisnode{%
4608   \ifnum\forestove{n children}>0
4609     \forest@pack@computetiers
4610     \forest@pack@level@nonuniform
4611     \forest@pack@aligntiers
4612     \forest@node@foreachchild{\forestoset{s}{0pt}}%
4613     \forest@pack@sibling@uniform@applyreversed
4614   \fi
4615 }

```

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

the same grow.

```

4616 \def\forest@pack@computegrowthuniformity{%
4617   \forest@node@foreachchild{\forest@pack@computegrowthuniformity}%
4618   \edef\forest@pack@cgu@uniformity{%
4619     \ifnum\forestove{n children}=0
4620     2\else 1\fi
4621   }%
4622   \foresttoget{grow}\forest@pack@cgu@parentgrow
4623   \forest@node@foreachchild{%
4624     \ifnum\forestove{uniform growth}=0
4625     \def\forest@pack@cgu@uniformity{0}%
4626     \else
4627     \ifnum\forestove{uniform growth}=1
4628     \ifnum\forestove{grow}=\forest@pack@cgu@parentgrow\relax\else
4629     \def\forest@pack@cgu@uniformity{0}%
4630     \fi
4631     \fi
4632   \fi
4633 }%
4634 \foresttoget{before packing node}\forest@temp@a
4635 \foresttoget{after packing node}\forest@temp@b
4636 \expandafter\expandafter\expandafter\ifstrempy\expandafter\expandafter\expandafter{\expandafter\forest@temp@a}
4637 \forestolet{uniform growth}\forest@pack@cgu@uniformity
4638 }{%
4639   \forestoset{uniform growth}{0}%
4640 }%
4641 }

```

Pack children in the level dimension in a uniform tree.

```

4642 \def\forest@pack@level@uniform{%
4643   \let\forest@plu@minchildl\relax
4644   \foresttoget{grow}\forest@plu@grow
4645   \forest@node@foreachchild{%
4646     \forest@node@getboundingrectangle@ls{\forest@plu@grow}%
4647     \advance\pgf@xa\forestove{1}\relax
4648     \ifx\forest@plu@minchildl\relax
4649     \edef\forest@plu@minchildl{\the\pgf@xa}%
4650     \else
4651     \ifdim\pgf@xa<\forest@plu@minchildl\relax
4652     \edef\forest@plu@minchildl{\the\pgf@xa}%
4653     \fi
4654   \fi
4655 }%
4656 \forest@node@getboundingrectangle@ls{\forest@plu@grow}%
4657 \pgfutil@tempdima=\pgf@xb\relax
4658 \advance\pgfutil@tempdima -\forest@plu@minchildl\relax
4659 \advance\pgfutil@tempdima \forestove{1 sep}\relax
4660 \ifdim\pgfutil@tempdima>0pt
4661   \forest@node@foreachchild{%
4662     \forestoeset{1}{\the\dimexpr\forestove{1}+\the\pgfutil@tempdima}%
4663   }%
4664   \fi
4665   \forest@node@foreachchild{%
4666     \ifnum\forestove{n children}>0
4667     \forest@pack@level@uniform
4668   \fi
4669 }%
4670 }

```

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

```

4671 \def\forest@pack@level@nonuniform{%

```

```

4672 \let\forest@plu@minchildl\relax
4673 \forestoget{grow}\forest@plu@grow
4674 \forest@node@foreachchild{%
4675   \forest@node@getedge{negative}{\forest@plu@grow}{\forest@plnu@negativechildedge}%
4676   \forest@node@getedge{positive}{\forest@plu@grow}{\forest@plnu@positivechildedge}%
4677   \def\forest@plnu@childedge{\forest@plnu@negativechildedge\forest@plnu@positivechildedge}%
4678   \forest@path@getboundingrectangle@ls\forest@plnu@childedge{\forest@plu@grow}%
4679   \advance\pgf@xa\forestove{1}\relax
4680   \ifx\forest@plu@minchildl\relax
4681     \edef\forest@plu@minchildl{\the\pgf@xa}%
4682   \else
4683     \ifdim\pgf@xa<\forest@plu@minchildl\relax
4684       \edef\forest@plu@minchildl{\the\pgf@xa}%
4685     \fi
4686   \fi
4687 }%
4688 \forest@node@getboundingrectangle@ls{\forest@plu@grow}%
4689 \pgfutil@tempdima=\pgf@xb\relax
4690 \advance\pgfutil@tempdima-\forest@plu@minchildl\relax
4691 \advance\pgfutil@tempdima\forestove{1 sep}\relax
4692 \ifdim\pgfutil@tempdima>0pt
4693   \forest@node@foreachchild{%
4694     \forestoeset{1}{\the\dimexpr\the\pgfutil@tempdima+\forestove{1}}%
4695   }%
4696 \fi
4697 }

```

Align tiers.

```

4698 \def\forest@pack@aligntiers{%
4699   \forestoget{grow}\forest@temp@parentgrow
4700   \forestoget{@tiers}\forest@temp@tiers
4701   \forlistloop\forest@pack@aligntier@\forest@temp@tiers
4702 }
4703 \def\forest@pack@aligntiers@ofsubtree{%
4704   \forest@node@foreach{\forest@pack@aligntiers}%
4705 }
4706 \def\forest@pack@aligntiers@computeabs1{%
4707   \forestoleto{abs@1}{1}%
4708   \forest@node@foreachdescendant{\forest@pack@aligntiers@computeabs1}%
4709 }
4710 \def\forest@pack@aligntiers@computeabs1@{%
4711   \forestoeset{abs@1}{\the\dimexpr\forestove{1}+\forestove{\forestove{@parent}}{abs@1}}%
4712 }
4713 \def\forest@pack@aligntier@#1{%
4714   \forest@pack@aligntiers@computeabs1
4715   \pgfutil@tempdima=-\maxdimen\relax
4716   \def\forest@temp@currenttier{#1}%
4717   \forest@node@foreach{%
4718     \forestoget{tier}\forest@temp@tier
4719     \ifx\forest@temp@currenttier\forest@temp@tier
4720       \ifdim\pgfutil@tempdima<\forestove{abs@1}\relax
4721         \pgfutil@tempdima=\forestove{abs@1}\relax
4722       \fi
4723     \fi
4724   }%
4725   \ifdim\pgfutil@tempdima=-\maxdimen\relax\else
4726     \forest@node@foreach{%
4727       \forestoget{tier}\forest@temp@tier
4728       \ifx\forest@temp@currenttier\forest@temp@tier
4729         \forestoeset{1}{\the\dimexpr\pgfutil@tempdima-\forestove{abs@1}+\forestove{1}}%
4730       \fi

```

```

4731 }%
4732 \fi
4733 }

```

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

```

4734 \def\forest@pack@sibling@uniform@recursive{%
4735   \forest@node@foreachchild{\forest@pack@sibling@uniform@recursive}%
4736   \forest@pack@sibling@uniform@applyreversed
4737 }

```

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

```

4738 \def\forest@pack@sibling@uniform@applyreversed{%
4739   \ifnum\forestove{n children}>1
4740     \ifnum\forestove{reversed}=0
4741       \forest@pack@sibling@uniform@main{first}{last}{next}{previous}%
4742     \else
4743       \forest@pack@sibling@uniform@main{last}{first}{previous}{next}%
4744     \fi
4745   \else
4746     \ifnum\forestove{n children}=1

```

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

```

4747     \csname forest@calign@\forestove{calign}\endcsname
4748   \fi
4749 \fi
4750 }

```

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

```

4751 \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.

```

4752   \forestoget{#1}\forest@child
4753   \edef\forest@marshal{%
4754     \noexpand\forest@fornode{\forestove{#1}}{%
4755       \noexpand\forest@node@getedge
4756       {positive}%
4757       {\forestove{grow}}}%
4758     \noexpand\forest@temp@edge
4759   }%
4760   }\forest@marshal
4761   \forest@pack@pgfpoint@child@position\forest@child
4762   \let\forest@previous@positive@edge\pgfutil@empty
4763   \forest@extendpath\forest@previous@positive@edge\forest@temp@edge{%
4764     \forest@get{\forest@child}{#3}\forest@child

```

Loop until the current child is the null node.

```

4765   \edef\forest@previous@child@s{0pt}%
4766   \safeloop
4767   \unless\ifnum\forest@child=0

```

Get the negative edge of the child.

```

4768     \edef\forest@temp{%
4769       \noexpand\forest@fornode{\forest@child}{%
4770         \noexpand\forest@node@getedge
4771         {negative}%
4772         {\forestove{grow}}}%
4773       \noexpand\forest@temp@edge
4774     }%
4775     }\forest@temp

```

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

```

4776   \forest@pack@pgfpoint@childposition\forest@child

```

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

```

4777   \let\forest@child@negative@edge\pgfutil@empty
4778   \forest@extendpath\forest@child@negative@edge\forest@temp@edge{}%

```

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

```

4779   \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!)

```

4780   \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`.

```

4781   \ifx\forest@csdistance\relax
4782     %\forest@reset{\forest@child}{s}{\forest@previous@child@s}%
4783   \else
4784     \advance\pgfutil@tempdimb-\forest@csdistance\relax
4785     \advance\pgfutil@tempdimb\forestove{s sep}\relax
4786     \forest@reset{\forest@child}{s}{\the\dimexpr\forestove{\forest@child}{s}-\forest@csdistance+\forestove{s
4787   \fi

```

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

```

4788   \ifdim\forestove{\forest@child}{s}<\forest@previous@child@s\relax
4789     \forest@reset{\forest@child}{s}{\forest@previous@child@s}%
4790   \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.

```

4791   \forest@get{\forest@child}{s}\forest@child@s
4792   \edef\forest@previous@child@s{\forest@child@s}%
4793   \edef\forest@temp{%
4794     \noexpand\forest@fornode{\forest@child}{%
4795       \noexpand\forest@node@getedge
4796         {positive}%
4797         {\forestove{grow}}}%
4798     \noexpand\forest@temp@edge
4799   }%
4800   }\forest@temp
4801   \forest@pack@pgfpoint@childposition\forest@child
4802   \forest@extendpath\forest@previous@positive@edge\forest@temp@edge{}%
4803   \forest@getpositivetightedgeofpath\forest@previous@positive@edge\forest@previous@positive@edge
4804   \forest@get{\forest@child}{#3}\forest@child
4805   \saferepeat

```

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

```

4806   \csname forest@calign@\forestove{calign}\endcsname
4807 }

```

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

```

4808 \def\forest@pack@pgfpoint@childposition#1{%
4809   {%
4810     \pgftransformreset
4811     \pgftransformrotate{\forestove{grow}}%
4812     \forest@fornode{#1}{%
4813       \pgfpointransformed{\pgfqpoint{\forestove{1}}{\forestove{s}}}%
4814     }%
4815   }%
4816 }

```

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

```

4817 \def\forest@pack@pgfpoint@positioninggrow#1{%
4818   {%
4819     \pgftransformreset
4820     \pgftransformrotate{#1}%
4821     \pgfpointtransformed{\pgfqpoint{\forestove{1}}{\forestove{s}}}%
4822   }%
4823 }

Child alignment.
4824 \def\forest@calign@s@shift#1{%
4825   \pgfutil@tempdima=#1\relax
4826   \forest@node@foreachchild{%
4827     \forestoeset{s}{\the\dimexpr\forestove{s}+\pgfutil@tempdima}%
4828   }%
4829 }
4830 \def\forest@calign@child{%
4831   \forest@calign@s@shift{-\forestOve{\forest@node@nornbarthchildid{\forestove{calign primary child}}}{s}}%
4832 }
4833 \csdef{forest@calign@child edge}{%
4834   {%
4835     \edef\forest@temp@child{\forest@node@nornbarthchildid{\forestove{calign primary child}}}%
4836     \pgftransformreset
4837     \pgftransformrotate{\forestove{grow}}%
4838     \pgfpointtransformed{\pgfqpoint{\forestOve{\forest@temp@child}{1}}{\forestOve{\forest@temp@child}{s}}}%
4839     \pgf@xa=\pgf@x\relax\pgf@ya=\pgf@y\relax
4840     \forest@Pointanchor{\forest@temp@child}{child anchor}%
4841     \advance\pgf@xa\pgf@x\relax\advance\pgf@ya\pgf@y\relax
4842     \forest@pointanchor{parent anchor}%
4843     \advance\pgf@xa-\pgf@x\relax\advance\pgf@ya-\pgf@y\relax
4844     \edef\forest@marshal{%
4845       \noexpand\pgftransformreset
4846       \noexpand\pgftransformrotate{-\forestove{grow}}%
4847       \noexpand\pgfpointtransformed{\noexpand\pgfqpoint{\the\pgf@xa}{\the\pgf@ya}}%
4848     }\forest@marshal
4849   }%
4850   \forest@calign@s@shift{\the\dimexpr-\the\pgf@y}%
4851 }
4852 \csdef{forest@calign@midpoint}{%
4853   \forest@calign@s@shift{\the\dimexpr Opt -%
4854     (\forestOve{\forest@node@nornbarthchildid{\forestove{calign primary child}}}{s}%
4855     +\forestOve{\forest@node@nornbarthchildid{\forestove{calign secondary child}}}{s}%
4856     )/2\relax
4857   }%
4858 }
4859 \csdef{forest@calign@edge midpoint}{%
4860   {%
4861     \edef\forest@temp@firstchild{\forest@node@nornbarthchildid{\forestove{calign primary child}}}%
4862     \edef\forest@temp@secondchild{\forest@node@nornbarthchildid{\forestove{calign secondary child}}}%
4863     \pgftransformreset
4864     \pgftransformrotate{\forestove{grow}}%
4865     \pgfpointtransformed{\pgfqpoint{\forestOve{\forest@temp@firstchild}{1}}{\forestOve{\forest@temp@firstchild}{s}}}%
4866     \pgf@xa=\pgf@x\relax\pgf@ya=\pgf@y\relax
4867     \forest@Pointanchor{\forest@temp@firstchild}{child anchor}%
4868     \advance\pgf@xa\pgf@x\relax\advance\pgf@ya\pgf@y\relax
4869     \edef\forest@marshal{%
4870       \noexpand\pgfpointtransformed{\noexpand\pgfqpoint{\forestOve{\forest@temp@secondchild}{1}}{\forestOve{\forest@temp@secondchild}{s}}}%
4871     }\forest@marshal
4872     \advance\pgf@xa\pgf@x\relax\advance\pgf@ya\pgf@y\relax
4873     \forest@Pointanchor{\forest@temp@secondchild}{child anchor}%
4874     \advance\pgf@xa\pgf@x\relax\advance\pgf@ya\pgf@y\relax

```

```

4875 \divide\pgf@xa2 \divide\pgf@ya2
4876 \forest@pointanchor{parent anchor}%
4877 \advance\pgf@xa-\pgf@x\relax\advance\pgf@ya-\pgf@y\relax
4878 \edef\forest@marshal{%
4879 \noexpand\pgftransformreset
4880 \noexpand\pgftransformrotate{-\forestove{grow}}}%
4881 \noexpand\pgfpointtransformed{\noexpand\pgfqpoint{\the\pgf@xa}{\the\pgf@ya}}%
4882 }\forest@marshal
4883 }%
4884 \forest@calign@s@shift{\the\dimexpr-\the\pgf@y}%
4885 }

```

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.

```

4886 \csdef{forest@calign@fixed angles}{%
4887 \ifnum\forestove{n children}>1
4888 \edef\forest@ca@first@child{\forest@node@nornbarthchildid{\forestove{calign primary child}}}%
4889 \edef\forest@ca@second@child{\forest@node@nornbarthchildid{\forestove{calign secondary child}}}%
4890 \ifnum\forestove{reversed}=1
4891 \let\forest@temp\forest@ca@first@child
4892 \let\forest@ca@first@child\forest@ca@second@child
4893 \let\forest@ca@second@child\forest@temp
4894 \fi
4895 \forest@get{\forest@ca@first@child}{l}\forest@ca@first@l
4896 \forest@get{\forest@ca@second@child}{l}\forest@ca@second@l
4897 \pgfmathsetlengthmacro\forest@ca@desired@s@distance{%
4898 tan(\forestove{calign secondary angle})*\forest@ca@second@l
4899 -tan(\forestove{calign primary angle})*\forest@ca@first@l
4900 }%
4901 \forest@get{\forest@ca@first@child}{s}\forest@ca@first@s
4902 \forest@get{\forest@ca@second@child}{s}\forest@ca@second@s
4903 \pgfmathsetlengthmacro\forest@ca@actual@s@distance{%
4904 \forest@ca@second@s-\forest@ca@first@s}%
4905 \ifdim\forest@ca@desired@s@distance>\forest@ca@actual@s@distance\relax
4906 \ifdim\forest@ca@actual@s@distance=0pt
4907 \pgfmathsetlength\pgfutil@tempdima{tan(\forestove{calign primary angle})*\forest@ca@second@l}%
4908 \pgfmathsetlength\pgfutil@tempdimb{\forest@ca@desired@s@distance/(\forestove{n children}-1)}%
4909 \forest@node@foreachchild{%
4910 \forestoeset{s}{\the\pgfutil@tempdima}%
4911 \advance\pgfutil@tempdima\pgfutil@tempdimb
4912 }%
4913 \def\forest@calign@anchor{0pt}%
4914 \else
4915 \pgfmathsetmacro\forest@ca@ratio{%
4916 \forest@ca@desired@s@distance/\forest@ca@actual@s@distance}%
4917 \forest@node@foreachchild{%
4918 \pgfmathsetlengthmacro\forest@temp{\forest@ca@ratio*\forestove{s}}%
4919 \forestolet{s}\forest@temp
4920 }%
4921 \pgfmathsetlengthmacro\forest@calign@anchor{%
4922 -tan(\forestove{calign primary angle})*\forest@ca@first@l}%
4923 \fi
4924 \else
4925 \ifdim\forest@ca@desired@s@distance<\forest@ca@actual@s@distance\relax
4926 \pgfmathsetlengthmacro\forest@ca@ratio{%
4927 \forest@ca@actual@s@distance/\forest@ca@desired@s@distance}%
4928 \forest@node@foreachchild{%
4929 \pgfmathsetlengthmacro\forest@temp{\forest@ca@ratio*\forestove{l}}%
4930 \forestolet{l}\forest@temp

```



```

4931     }%
4932     \forestOget{\forest@ca@first@child}{1}\forest@ca@first@l
4933     \pgfmathsetlengthmacro\forest@calign@anchor{%
4934         -tan(\forestove{calign primary angle})*\forest@ca@first@l}%
4935     \fi
4936     \fi
4937     \forest@calign@s@shift{-\forest@calign@anchor}%
4938     \fi
4939 }
4940 \csdef{forest@calign@fixed edge angles}{%
4941     \ifnum\forestove{n children}>1
4942         \edef\forest@ca@first@child{\forest@node@nornbarthchildid{\forestove{calign primary child}}}%
4943         \edef\forest@ca@second@child{\forest@node@nornbarthchildid{\forestove{calign secondary child}}}%
4944         \ifnum\forestove{reversed}=1
4945             \let\forest@temp\forest@ca@first@child
4946             \let\forest@ca@first@child\forest@ca@second@child
4947             \let\forest@ca@second@child\forest@temp
4948         \fi
4949         \forestOget{\forest@ca@first@child}{1}\forest@ca@first@l
4950         \forestOget{\forest@ca@second@child}{1}\forest@ca@second@l
4951         \forest@pointanchor{parent anchor}%
4952         \edef\forest@ca@parent@anchor@s{\the\pgf@x}%
4953         \edef\forest@ca@parent@anchor@l{\the\pgf@y}%
4954         \forest@Pointanchor{\forest@ca@first@child}{child anchor}%
4955         \edef\forest@ca@first@child@anchor@s{\the\pgf@x}%
4956         \edef\forest@ca@first@child@anchor@l{\the\pgf@y}%
4957         \forest@Pointanchor{\forest@ca@second@child}{child anchor}%
4958         \edef\forest@ca@second@child@anchor@s{\the\pgf@x}%
4959         \edef\forest@ca@second@child@anchor@l{\the\pgf@y}%
4960         \pgfmathsetlengthmacro\forest@ca@desired@second@edge@s{tan(\forestove{calign secondary angle})*%
4961             (\forest@ca@second@l-\forest@ca@second@child@anchor@l+\forest@ca@parent@anchor@l)}%
4962         \pgfmathsetlengthmacro\forest@ca@desired@first@edge@s{tan(\forestove{calign primary angle})*%
4963             (\forest@ca@first@l-\forest@ca@first@child@anchor@l+\forest@ca@parent@anchor@l)}
4964         \pgfmathsetlengthmacro\forest@ca@desired@s@distance{\forest@ca@desired@second@edge@s-\forest@ca@desired@f
4965         \forestOget{\forest@ca@first@child}{s}\forest@ca@first@s
4966         \forestOget{\forest@ca@second@child}{s}\forest@ca@second@s
4967         \pgfmathsetlengthmacro\forest@ca@actual@s@distance{%
4968             \forest@ca@second@s+\forest@ca@second@child@anchor@s
4969             -\forest@ca@first@s-\forest@ca@first@child@anchor@s}%
4970         \ifdim\forest@ca@desired@s@distance>\forest@ca@actual@s@distance\relax
4971             \ifdim\forest@ca@actual@s@distance=0pt
4972                 \forestoget{n children}\forest@temp@n@children
4973                 \forest@node@foreachchild{%
4974                     \forest@pointanchor{child anchor}%
4975                     \edef\forest@temp@child@anchor@s{\the\pgf@x}%
4976                     \pgfmathsetlengthmacro\forest@temp{%
4977                         \forest@ca@desired@first@edge@s+(\forestove{n}-1)*\forest@ca@desired@s@distance/(\forest@temp@n@
4978                     \forestolet{s}\forest@temp
4979                 }%
4980                 \def\forest@calign@anchor{0pt}%
4981             \else
4982                 \pgfmathsetmacro\forest@ca@ratio{%
4983                     \forest@ca@desired@s@distance/\forest@ca@actual@s@distance}%
4984                 \forest@node@foreachchild{%
4985                     \forest@pointanchor{child anchor}%
4986                     \edef\forest@temp@child@anchor@s{\the\pgf@x}%
4987                     \pgfmathsetlengthmacro\forest@temp{%
4988                         \forest@ca@ratio*(%
4989                             \forestove{s}-\forest@ca@first@s
4990                             +\forest@temp@child@anchor@s-\forest@ca@first@child@anchor@s)%
4991                     +\forest@ca@first@s

```

```

4992         +\forest@ca@first@child@anchor@s-\forest@temp@child@anchor@s}%
4993     \forestolet{s}\forest@temp
4994 }%
4995     \pgfmathsetlengthmacro\forest@calign@anchor{%
4996         -tan(\forestove{calign primary angle})*(\forest@ca@first@l-\forest@ca@first@child@anchor@l+\forest@
4997         +\forest@ca@first@child@anchor@s-\forest@ca@parent@anchor@s
4998     }%
4999     \fi
5000 \else
5001     \ifdim\forest@ca@desired@s@distance<\forest@ca@actual@s@distance\relax
5002     \pgfmathsetlengthmacro\forest@ca@ratio{%
5003         \forest@ca@actual@s@distance/\forest@ca@desired@s@distance}%
5004     \forest@node@foreachchild{%
5005         \forest@pointanchor{child anchor}%
5006         \edef\forest@temp@child@anchor@l{\the\pgf@y}%
5007         \pgfmathsetlengthmacro\forest@temp{%
5008             \forest@ca@ratio*(%
5009                 \forestove{l}+\forest@ca@parent@anchor@l-\forest@temp@child@anchor@l)
5010                 -\forest@ca@parent@anchor@l+\forest@temp@child@anchor@l}%
5011         \forestolet{l}\forest@temp
5012     }%
5013     \forest@get{\forest@ca@first@child}{l}\forest@ca@first@l
5014     \pgfmathsetlengthmacro\forest@calign@anchor{%
5015         -tan(\forestove{calign primary angle})*(\forest@ca@first@l+\forest@ca@parent@anchor@l-\forest@temp@
5016         +\forest@ca@first@child@anchor@s-\forest@ca@parent@anchor@s
5017     }%
5018     \fi
5019     \fi
5020     \forest@calign@s@shift{-\forest@calign@anchor}%
5021 \fi
5022 }

```

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.

```

5023 \def\forest@node@getedge#1#2#3{%
5024     \forestoget{#1@edge@#2}#3%
5025     \ifx#3\relax
5026         \forest@node@foreachchild{%
5027             \forest@node@getedge{#1}{#2}{\forest@temp@edge}%
5028         }%
5029         \forest@forthis{\forest@node@getedges{#2}}%
5030         \forestoget{#1@edge@#2}#3%
5031     \fi
5032 }

```

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.

```

5033 \def\forest@node@getedges#1{%

```

Run the computation in a `TeX` group.

```

5034     {%

```

Setup the grow line.

```

5035         \forest@setupgrowline{#1}%

```

Get the edge of the node itself.

```

5036         \ifnum\forestove{ignore}=0
5037         \forestoget{@boundary}\forest@node@boundary
5038         \else
5039             \def\forest@node@boundary{}%
5040         \fi

```

```

5041 \csname forest@getboth\forestove{fit}edgesofpath\endcsname
5042 \forest@node@boundary\forest@negative@node@edge\forest@positive@node@edge
5043 \forest@tolet{negative@edge@#1}\forest@negative@node@edge
5044 \forest@tolet{positive@edge@#1}\forest@positive@node@edge

Add the edges of the children.
5045 \get@edges@merge{negative}{#1}%
5046 \get@edges@merge{positive}{#1}%
5047 }%%
5048 }

Merge the #1 (=negative or positive) edge of the node with #1 edges of the children. #2 = grow angle.
5049 \def\get@edges@merge#1#2{%
5050 \ifnum\forestove{n children}>0
5051 \forest@toget{#1@edge@#2}\forest@node@edge

Remember the node's parent anchor and add it to the path (for breaking).
5052 \forest@pointanchor{parent anchor}%
5053 \edef\forest@getedge@pa@1{\the\pgf@x}%
5054 \edef\forest@getedge@pa@s{\the\pgf@y}%
5055 \eappto\forest@node@edge{\noexpand\pgfsyssoftpath@movetotoken{\forest@getedge@pa@1}{\forest@getedge@pa@s}%

Switch to this node's (l,s) coordinate system (origin at the node's anchor).
5056 \pgfgettransform\forest@temp@transform
5057 \pgftransformreset
5058 \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.
5059 \def\forest@all@edges{%
5060 \forest@node@foreachchild{%
5061 \forest@toget{#1@edge@#2}\forest@temp@edge
5062 \pgfpointtransformed{\pgfpoint{\forestove{1}}{\forestove{s}}}%
5063 \forest@extendpath\forest@node@edge\forest@temp@edge}%
5064 \ifnum\forestove{ignore edge}=0
5065 \pgfpointadd
5066 {\pgfpointtransformed{\pgfpoint{\forestove{1}}{\forestove{s}}}%
5067 {\forest@pointanchor{child anchor}}}%
5068 \pgfgetlastxy{\forest@getedge@ca@1}{\forest@getedge@ca@s}%
5069 \eappto\forest@all@edges{%
5070 \noexpand\pgfsyssoftpath@movetotoken{\forest@getedge@pa@1}{\forest@getedge@pa@s}%
5071 \noexpand\pgfsyssoftpath@linetotoken{\forest@getedge@ca@1}{\forest@getedge@ca@s}%
5072 }%
5073 % this deals with potential overlap of the edges:
5074 \eappto\forest@node@edge{\noexpand\pgfsyssoftpath@movetotoken{\forest@getedge@ca@1}{\forest@getedge@ca@s}%
5075 \fi
5076 }%
5077 \ifdefempty{\forest@all@edges}{}%
5078 \pgfintersectionofpaths{\pgfsetpath\forest@all@edges}{\pgfsetpath\forest@node@edge}%
5079 \def\forest@edgenode@intersections{%
5080 \forest@merge@intersectionloop
5081 \eappto\forest@node@edge{\expandonce{\forest@all@edges}\expandonce{\forest@edgenode@intersections}}%
5082 }%
5083 \pgfsettransform\forest@temp@transform

Process the path into an edge and store the edge.
5084 \csname forest@get#1\forestove{fit}edgeofpath\endcsname\forest@node@edge\forest@node@edge
5085 \forest@tolet{#1@edge@#2}\forest@node@edge
5086 \fi
5087 }

```

```

5088 %\newloop\forest@merge@loop
5089 \def\forest@merge@intersectionloop{%
5090   \c@pgf@counta=0
5091   \forest@loop
5092   \ifnum\c@pgf@counta<\pgfintersectionsolutions\relax
5093     \advance\c@pgf@counta1
5094     \pgfpointintersectionsolution{\the\c@pgf@counta}%
5095     \eappto\forest@edgenode@intersections{\noexpand\pgfsyssoftpath@movetotoken
5096       {\the\pgf@x}{\the\pgf@y}}%
5097   \forest@repeat
5098 }

```

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

```

5099 \def\forest@node@getboundingrectangle@ls#1{%
5100   \forestoget{@boundary}\forest@node@boundary
5101   \forest@path@getboundingrectangle@ls\forest@node@boundary{#1}%
5102 }

```

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).

```

5103 \def\forest@pgfpathtransformed#1{%
5104   \forest@save@pgfsyssoftpath@tokendefs
5105   \let\pgfsyssoftpath@movetotoken\forest@pgfpathtransformed@moveto
5106   \let\pgfsyssoftpath@linetotoken\forest@pgfpathtransformed@lineto
5107   \pgfsyssoftpath@setcurrentpath\pgfutil@empty
5108   #1%
5109   \forest@restore@pgfsyssoftpath@tokendefs
5110 }
5111 \def\forest@pgfpathtransformed@moveto#1#2{%
5112   \forest@pgfpathtransformed@op\pgfsyssoftpath@moveto{#1}{#2}%
5113 }
5114 \def\forest@pgfpathtransformed@lineto#1#2{%
5115   \forest@pgfpathtransformed@op\pgfsyssoftpath@lineto{#1}{#2}%
5116 }
5117 \def\forest@pgfpathtransformed@op#1#2#3{%
5118   \pgfpointtransformed{\pgfpoint{#2}{#3}}%
5119   \edef\forest@temp{%
5120     \noexpand#1{\the\pgf@x}{\the\pgf@y}%
5121   }%
5122   \forest@temp
5123 }

```

7.2.1 Tiers

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

```

5124 \def\forest@pack@computetiers{%
5125   {%
5126     \forest@pack@tiers@getalltiersinsubtree
5127     \forest@pack@tiers@computetierhierarchy
5128     \forest@pack@tiers@findcontainers
5129     \forest@pack@tiers@raisecontainers
5130     \forest@pack@tiers@computeprocessingorder
5131     \gdef\forest@smuggle{%
5132       \forest@pack@tiers@write
5133     }%
5134     \forest@node@foreach{\forestoset{@tiers}{}}%
5135   \forest@smuggle
5136 }

```

Puts all tiers contained in the subtree into attribute tiers.

```

5137 \def\forest@pack@tiers@getalltiersinsubtree{%
5138   \ifnum\forestove{n children}>0

```

```

5139 \forest@node@foreachchild{\forest@pack@tiers@getalltiersinsubtree}%
5140 \fi
5141 \forest@toget{tier}\forest@temp@mytier
5142 \def\forest@temp@mytiers{}%
5143 \ifdefempty\forest@temp@mytier{}-%
5144 \list@add\forest@temp@mytiers\forest@temp@mytier
5145 }%
5146 \ifnum\forest@ove{n children}>0
5147 \forest@node@foreachchild{%
5148 \forest@toget{tiers}\forest@temp@tiers
5149 \forlistloop\forest@pack@tiers@forhandlerA\forest@temp@tiers
5150 }%
5151 \fi
5152 \forest@tolet{tiers}\forest@temp@mytiers
5153 }
5154 \def\forest@pack@tiers@forhandlerA#1{%
5155 \ifinlist{#1}\forest@temp@mytiers{}-%
5156 \list@add\forest@temp@mytiers{#1}%
5157 }%
5158 }

```

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.

```

5159 \def\forest@pack@tiers@computetierhierarchy{%
5160 \def\forest@tiers@ancestors{}%
5161 \forest@toget{tiers}\forest@temp@mytiers
5162 \forlistloop\forest@pack@tiers@cth@init\forest@temp@mytiers
5163 \forest@pack@tiers@computetierhierarchy@
5164 }
5165 \def\forest@pack@tiers@cth@init#1{%
5166 \csdef{forest@tiers@higher@#1}{}%
5167 \csdef{forest@tiers@lower@#1}{}%
5168 }
5169 \def\forest@pack@tiers@computetierhierarchy@{%
5170 \forest@toget{tier}\forest@temp@mytier
5171 \ifdefempty\forest@temp@mytier{}-%
5172 \forlistloop\forest@pack@tiers@forhandlerB\forest@tiers@ancestors
5173 \list@add\forest@tiers@ancestors\forest@temp@mytier
5174 }%
5175 \forest@node@foreachchild{%
5176 \forest@pack@tiers@computetierhierarchy@
5177 }%
5178 \forest@toget{tier}\forest@temp@mytier
5179 \ifdefempty\forest@temp@mytier{}-%
5180 \forest@list@del\forest@tiers@ancestors\forest@temp@mytier
5181 }%
5182 }
5183 \def\forest@pack@tiers@forhandlerB#1{%
5184 \def\forest@temp@tier{#1}%
5185 \ifx\forest@temp@tier\forest@temp@mytier
5186 \PackageError{forest}{Circular tier hierarchy (tier \forest@temp@mytier)}{ }%
5187 \fi
5188 \ifinlistcs{#1}{forest@tiers@higher@\forest@temp@mytier}{ }-%
5189 \listcs@add{forest@tiers@higher@\forest@temp@mytier}{#1}%
5190 \xifinlistcs\forest@temp@mytier{forest@tiers@lower@#1}{ }-%
5191 \listcs@eadd{forest@tiers@lower@#1}{\forest@temp@mytier}%
5192 }
5193 \def\forest@pack@tiers@findcontainers{%
5194 \forest@toget{tiers}\forest@temp@tiers
5195 \forlistloop\forest@pack@tiers@findcontainer\forest@temp@tiers
5196 }

```

```

5197 \def\forest@pack@tiers@findcontainer#1{%
5198   \def\forest@temp@tier{#1}%
5199   \foresttoget{tier}\forest@temp@mytier
5200   \ifx\forest@temp@tier\forest@temp@mytier
5201     \csedef{forest@tiers@container@#1}{\forest@cn}%
5202   \else\@escapeif{%
5203     \forest@pack@tiers@findcontainerA{#1}%
5204   }\fi%
5205 }
5206 \def\forest@pack@tiers@findcontainerA#1{%
5207   \c@pgf@counta=0
5208   \forest@node@foreachchild{%
5209     \foresttoget{tiers}\forest@temp@tiers
5210     \ifinlist{#1}\forest@temp@tiers{%
5211       \advance\c@pgf@counta 1
5212       \let\forest@temp@child\forest@cn
5213     }{}%
5214   }%
5215   \ifnum\c@pgf@counta>1
5216     \csedef{forest@tiers@container@#1}{\forest@cn}%
5217   \else\@escapeif{ surely =1
5218     \forest@fornode{\forest@temp@child}{%
5219       \forest@pack@tiers@findcontainer{#1}%
5220     }%
5221   }\fi
5222 }
5223 \def\forest@pack@tiers@raisecontainers{%
5224   \foresttoget{tiers}\forest@temp@mytiers
5225   \forlistloop\forest@pack@tiers@rc@forhandlerA\forest@temp@mytiers
5226 }
5227 \def\forest@pack@tiers@rc@forhandlerA#1{%
5228   \edef\forest@tiers@temptier{#1}%
5229   \letcs\forest@tiers@containernodeoftier{forest@tiers@container@#1}%
5230   \letcs\forest@temp@lowertiers{forest@tiers@lower@#1}%
5231   \forlistloop\forest@pack@tiers@rc@forhandlerB\forest@temp@lowertiers
5232 }
5233 \def\forest@pack@tiers@rc@forhandlerB#1{%
5234   \letcs\forest@tiers@containernodeoflowertier{forest@tiers@container@#1}%
5235   \forest@get{\forest@tiers@containernodeoflowertier}{content}\lowercontent
5236   \forest@get{\forest@tiers@containernodeoftier}{content}\uppercontent
5237   \forest@fornode{\forest@tiers@containernodeoflowertier}{%
5238     \forest@ifancestorof
5239       {\forest@tiers@containernodeoftier}
5240       {\csletcs{forest@tiers@container\forest@tiers@temptier}{forest@tiers@container@#1}}%
5241     {}%
5242   }%
5243 }
5244 \def\forest@pack@tiers@computeprocessingorder{%
5245   \def\forest@tiers@processingorder{%
5246     \foresttoget{tiers}\forest@tiers@cpo@tierstodo
5247     \safeloop
5248     \ifdefempty\forest@tiers@cpo@tierstodo{\forest@tempfalse}{\forest@temptrue}%
5249     \ifforest@temp
5250       \def\forest@tiers@cpo@tiersremaining{%
5251         \def\forest@tiers@cpo@tiersindependent{%
5252           \forlistloop\forest@pack@tiers@cpo@forhandlerA\forest@tiers@cpo@tierstodo
5253           \ifdefempty\forest@tiers@cpo@tiersindependent{%
5254             \PackageError{forest}{Circular tiers!}{}}{}%
5255           \forlistloop\forest@pack@tiers@cpo@forhandlerB\forest@tiers@cpo@tiersremaining
5256           \let\forest@tiers@cpo@tierstodo\forest@tiers@cpo@tiersremaining
5257         \saferepeat

```

```

5258 }
5259 \def\forest@pack@tiers@cpo@forhandlerA#1{%
5260   \ifcempty{forest@tiers@higher@#1}{%
5261     \listadd\forest@tiers@cpo@tiersindependent{#1}%
5262     \listadd\forest@tiers@processingorder{#1}%
5263   }{%
5264     \listadd\forest@tiers@cpo@tiersremaining{#1}%
5265   }%
5266 }
5267 \def\forest@pack@tiers@cpo@forhandlerB#1{%
5268   \def\forest@pack@tiers@cpo@aremainingtier{#1}%
5269   \forlistloop\forest@pack@tiers@cpo@forhandlerC\forest@tiers@cpo@tiersindependent
5270 }
5271 \def\forest@pack@tiers@cpo@forhandlerC#1{%
5272   \ifinlistcs{#1}{forest@tiers@higher@\forest@pack@tiers@cpo@aremainingtier}{%
5273     \forest@listcsdel{forest@tiers@higher@\forest@pack@tiers@cpo@aremainingtier}{#1}%
5274   }{}%
5275 }
5276 \def\forest@pack@tiers@write{%
5277   \forlistloop\forest@pack@tiers@write@forhandler\forest@tiers@processingorder
5278 }
5279 \def\forest@pack@tiers@write@forhandler#1{%
5280   \forest@fornode{\csname forest@tiers@container@#1\endcsname}{%
5281     \forest@pack@tiers@check{#1}%
5282   }%
5283   \xappto\forest@smuggle{%
5284     \noexpand\listadd
5285     \forest@om{\csname forest@tiers@container@#1\endcsname}{@tiers}%
5286     {#1}%
5287   }%
5288 }
5289 % checks if the tier is compatible with growth changes and calign=node/edge angle
5290 \def\forest@pack@tiers@check#1{%
5291   \def\forest@temp@currenttier{#1}%
5292   \forest@node@foreachdescendant{%
5293     \ifnum\forestove{grow}=\forestove{\forestove{@parent}}{grow}
5294     \else
5295       \forest@pack@tiers@check@grow
5296     \fi
5297     \ifnum\forestove{n children}>1
5298       \forest@toget{calign}\forest@temp
5299       \ifx\forest@temp\forest@pack@tiers@check@nodeangle
5300         \forest@pack@tiers@check@calign
5301       \fi
5302       \ifx\forest@temp\forest@pack@tiers@check@edgeangle
5303         \forest@pack@tiers@check@calign
5304       \fi
5305     \fi
5306   }%
5307 }
5308 \def\forest@pack@tiers@check@nodeangle{node angle}%
5309 \def\forest@pack@tiers@check@edgeangle{edge angle}%
5310 \def\forest@pack@tiers@check@grow{%
5311   \forest@toget{content}\forest@temp@content
5312   \let\forest@temp@currentnode\forest@cn
5313   \forest@node@foreachdescendant{%
5314     \forest@toget{tier}\forest@temp
5315     \ifx\forest@temp@currenttier\forest@temp
5316       \forest@pack@tiers@check@grow@error
5317     \fi
5318   }%

```

```

5319 }
5320 \def\forest@pack@tiers@check@grow@error{%
5321   \PackageError{forest}{Tree growth direction changes in node \forest@temp@currentnode\space
5322     (content: \forest@temp@content), while tier '\forest@temp' is specified for nodes both
5323     out- and inside the subtree rooted in node \forest@temp@currentnode. This will not work.}{}%
5324 }
5325 \def\forest@pack@tiers@check@calign{%
5326   \forest@node@foreachchild{%
5327     \forest@toget{tier}\forest@temp
5328     \ifx\forest@temp@currenttier\forest@temp
5329       \forest@pack@tiers@check@calign@warning
5330     \fi
5331   }%
5332 }
5333 \def\forest@pack@tiers@check@calign@warning{%
5334   \PackageWarning{forest}{Potential option conflict: node \forest@temp@parent (content:
5335     '\forest@temp@parent') was given 'calign=\forest@temp@calign', while its
5336     child \forest@temp@child (content: '\forest@temp@child') was given 'tier=\forest@temp@tier'.
5337     The parent's 'calign' will only work if the child was the lowest node on its tier before the
5338     alignment.}%
5339 }

```

7.2.2 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.

```

5340 \pgfdeclaregenericanchor{forest@compute@node@boundary}{%
5341   \letcs\forest@temp@boundary@macro\forest@compute@node@boundary@#1}%
5342   \ifcsname forest@compute@node@boundary@#1\endcsname
5343     \csname forest@compute@node@boundary@#1\endcsname
5344   \else
5345     \forest@compute@node@boundary@rectangle
5346   \fi
5347   \pgfsyssoftpath@getcurrentpath\forest@temp
5348   \global\let\forest@global@boundary\forest@temp
5349 }
5350 \def\forest@mt#1{%
5351   \expandafter\pgfpointanchor\expandafter{\pgfreferencednodename}{#1}%
5352   \pgfsyssoftpath@moveto{\the\pgf@x}{\the\pgf@y}%
5353 }%
5354 \def\forest@lt#1{%
5355   \expandafter\pgfpointanchor\expandafter{\pgfreferencednodename}{#1}%
5356   \pgfsyssoftpath@lineto{\the\pgf@x}{\the\pgf@y}%
5357 }%
5358 \def\forest@compute@node@boundary@coordinate{%
5359   \forest@mt{center}%
5360 }
5361 \def\forest@compute@node@boundary@circle{%
5362   \forest@mt{east}%
5363   \forest@lt{north east}%
5364   \forest@lt{north}%
5365   \forest@lt{north west}%
5366   \forest@lt{west}%
5367   \forest@lt{south west}%
5368   \forest@lt{south}%
5369   \forest@lt{south east}%
5370   \forest@lt{east}%
5371 }
5372 \def\forest@compute@node@boundary@rectangle{%
5373   \forest@mt{south west}%

```



```

5374 \forest@lt{south east}%
5375 \forest@lt{north east}%
5376 \forest@lt{north west}%
5377 \forest@lt{south west}%
5378 }
5379 \def\forest@compute@node@boundary@diamond{%
5380 \forest@mt{east}%
5381 \forest@lt{north}%
5382 \forest@lt{west}%
5383 \forest@lt{south}%
5384 \forest@lt{east}%
5385 }
5386 \let\forest@compute@node@boundary@ellipse\forest@compute@node@boundary@circle
5387 \def\forest@compute@node@boundary@trapezium{%
5388 \forest@mt{top right corner}%
5389 \forest@lt{top left corner}%
5390 \forest@lt{bottom left corner}%
5391 \forest@lt{bottom right corner}%
5392 \forest@lt{top right corner}%
5393 }
5394 \def\forest@compute@node@boundary@semicircle{%
5395 \forest@mt{arc start}%
5396 \forest@lt{north}%
5397 \forest@lt{east}%
5398 \forest@lt{north east}%
5399 \forest@lt{apex}%
5400 \forest@lt{north west}%
5401 \forest@lt{west}%
5402 \forest@lt{arc end}%
5403 \forest@lt{arc start}%
5404 }
5405 %\newloop\forest@computenodeboundary@loop
5406 \csdef{forest@compute@node@boundary@regular polygon}{%
5407 \forest@mt{corner 1}%
5408 \c@pgf@counta=\sides\relax
5409 \forest@loop
5410 \ifnum\c@pgf@counta>0
5411 \forest@lt{corner \the\c@pgf@counta}%
5412 \advance\c@pgf@counta-1
5413 \forest@repeat
5414 }%
5415 \def\forest@compute@node@boundary@star{%
5416 \forest@mt{outer point 1}%
5417 \c@pgf@counta=\totalstarpoints\relax
5418 \divide\c@pgf@counta2
5419 \forest@loop
5420 \ifnum\c@pgf@counta>0
5421 \forest@lt{inner point \the\c@pgf@counta}%
5422 \forest@lt{outer point \the\c@pgf@counta}%
5423 \advance\c@pgf@counta-1
5424 \forest@repeat
5425 }%
5426 \csdef{forest@compute@node@boundary@isosceles triangle}{%
5427 \forest@mt{apex}%
5428 \forest@lt{left corner}%
5429 \forest@lt{right corner}%
5430 \forest@lt{apex}%
5431 }
5432 \def\forest@compute@node@boundary@kite{%
5433 \forest@mt{upper vertex}%
5434 \forest@lt{left vertex}%

```

```

5435 \forest@lt{lower vertex}%
5436 \forest@lt{right vertex}%
5437 \forest@lt{upper vertex}%
5438 }
5439 \def\forest@compute@node@boundary@dart{%
5440 \forest@mt{tip}%
5441 \forest@lt{left tail}%
5442 \forest@lt{tail center}%
5443 \forest@lt{right tail}%
5444 \forest@lt{tip}%
5445 }
5446 \csdef{forest@compute@node@boundary@circular sector}{%
5447 \forest@mt{sector center}%
5448 \forest@lt{arc start}%
5449 \forest@lt{arc center}%
5450 \forest@lt{arc end}%
5451 \forest@lt{sector center}%
5452 }
5453 \def\forest@compute@node@boundary@cylinder{%
5454 \forest@mt{top}%
5455 \forest@lt{after top}%
5456 \forest@lt{before bottom}%
5457 \forest@lt{bottom}%
5458 \forest@lt{after bottom}%
5459 \forest@lt{before top}%
5460 \forest@lt{top}%
5461 }
5462 \cslet{forest@compute@node@boundary@forbidden sign}\forest@compute@node@boundary@circle
5463 \cslet{forest@compute@node@boundary@magnifying glass}\forest@compute@node@boundary@circle
5464 \def\forest@compute@node@boundary@cloud{%
5465 \getradii
5466 \forest@mt{puff 1}%
5467 \c@pgf@counta=\puffs\relax
5468 \forest@loop
5469 \ifnum\c@pgf@counta>0
5470 \forest@lt{puff \the\c@pgf@counta}%
5471 \advance\c@pgf@counta-1
5472 \forest@repeat
5473 }
5474 \def\forest@compute@node@boundary@starburst{
5475 \calculatestarburstpoints
5476 \forest@mt{outer point 1}%
5477 \c@pgf@counta=\totalpoints\relax
5478 \divide\c@pgf@counta2
5479 \forest@loop
5480 \ifnum\c@pgf@counta>0
5481 \forest@lt{inner point \the\c@pgf@counta}%
5482 \forest@lt{outer point \the\c@pgf@counta}%
5483 \advance\c@pgf@counta-1
5484 \forest@repeat
5485 }%
5486 \def\forest@compute@node@boundary@signal{%
5487 \forest@mt{east}%
5488 \forest@lt{south east}%
5489 \forest@lt{south west}%
5490 \forest@lt{west}%
5491 \forest@lt{north west}%
5492 \forest@lt{north east}%
5493 \forest@lt{east}%
5494 }
5495 \def\forest@compute@node@boundary@tape{%

```

```

5496 \forest@mt{north east}%
5497 \forest@lt{60}%
5498 \forest@lt{north}%
5500 \forest@lt{north west}%
5501 \forest@lt{south west}%
5502 \forest@lt{240}%
5503 \forest@lt{south}%
5504 \forest@lt{310}%
5505 \forest@lt{south east}%
5506 \forest@lt{north east}%
5507 }
5508 \csdef{forest@compute@node@boundary@single arrow}{%
5509 \forest@mt{tip}%
5510 \forest@lt{after tip}%
5511 \forest@lt{after head}%
5512 \forest@lt{before tail}%
5513 \forest@lt{after tail}%
5514 \forest@lt{before head}%
5515 \forest@lt{before tip}%
5516 \forest@lt{tip}%
5517 }
5518 \csdef{forest@compute@node@boundary@double arrow}{%
5519 \forest@mt{tip 1}%
5520 \forest@lt{after tip 1}%
5521 \forest@lt{after head 1}%
5522 \forest@lt{before head 2}%
5523 \forest@lt{before tip 2}%
5524 \forest@mt{tip 2}%
5525 \forest@lt{after tip 2}%
5526 \forest@lt{after head 2}%
5527 \forest@lt{before head 1}%
5528 \forest@lt{before tip 1}%
5529 \forest@lt{tip 1}%
5530 }
5531 \csdef{forest@compute@node@boundary@arrow box}{%
5532 \forest@mt{before north arrow}%
5533 \forest@lt{before north arrow head}%
5534 \forest@lt{before north arrow tip}%
5535 \forest@lt{north arrow tip}%
5536 \forest@lt{after north arrow tip}%
5537 \forest@lt{after north arrow head}%
5538 \forest@lt{after north arrow}%
5539 \forest@lt{north east}%
5540 \forest@lt{before east arrow}%
5541 \forest@lt{before east arrow head}%
5542 \forest@lt{before east arrow tip}%
5543 \forest@lt{east arrow tip}%
5544 \forest@lt{after east arrow tip}%
5545 \forest@lt{after east arrow head}%
5546 \forest@lt{after east arrow}%
5547 \forest@lt{south east}%
5548 \forest@lt{before south arrow}%
5549 \forest@lt{before south arrow head}%
5550 \forest@lt{before south arrow tip}%
5551 \forest@lt{south arrow tip}%
5552 \forest@lt{after south arrow tip}%
5553 \forest@lt{after south arrow head}%
5554 \forest@lt{after south arrow}%
5555 \forest@lt{south west}%
5556 \forest@lt{before west arrow}%

```

```

5557 \forest@lt{before west arrow head}%
5558 \forest@lt{before west arrow tip}%
5559 \forest@lt{west arrow tip}%
5560 \forest@lt{after west arrow tip}%
5561 \forest@lt{after west arrow head}%
5562 \forest@lt{after west arrow}%
5563 \forest@lt{north west}%
5564 \forest@lt{before north arrow}%
5565 }
5566 \cslet{forest@compute@node@boundary@circle split}\forest@compute@node@boundary@circle
5567 \cslet{forest@compute@node@boundary@circle solidus}\forest@compute@node@boundary@circle
5568 \cslet{forest@compute@node@boundary@ellipse split}\forest@compute@node@boundary@ellipse
5569 \cslet{forest@compute@node@boundary@rectangle split}\forest@compute@node@boundary@rectangle
5570 \def\forest@compute@node@boundary@@callout{%
5571 \beforecalloutpointer
5572 \pgfsyssoftpath@moveto{\the\pgf@x}{\the\pgf@y}%
5573 \calloutpointeranchor
5574 \pgfsyssoftpath@lineto{\the\pgf@x}{\the\pgf@y}%
5575 \aftercalloutpointer
5576 \pgfsyssoftpath@lineto{\the\pgf@x}{\the\pgf@y}%
5577 }
5578 \csdef{forest@compute@node@boundary@rectangle callout}{%
5579 \forest@compute@node@boundary@rectangle
5580 \rectanglecalloutpoints
5581 \forest@compute@node@boundary@@callout
5582 }
5583 \csdef{forest@compute@node@boundary@ellipse callout}{%
5584 \forest@compute@node@boundary@ellipse
5585 \ellipsecalloutpoints
5586 \forest@compute@node@boundary@@callout
5587 }
5588 \csdef{forest@compute@node@boundary@cloud callout}{%
5589 \forest@compute@node@boundary@cloud
5590 % at least a first approx...
5591 \forest@mt{center}%
5592 \forest@lt{pointer}%
5593 }%
5594 \csdef{forest@compute@node@boundary@cross out}{%
5595 \forest@mt{south east}%
5596 \forest@lt{north west}%
5597 \forest@mt{south west}%
5598 \forest@lt{north east}%
5599 }%
5600 \csdef{forest@compute@node@boundary@strike out}{%
5601 \forest@mt{north east}%
5602 \forest@lt{south west}%
5603 }%
5604 \csdef{forest@compute@node@boundary@rounded rectangle}{%
5605 \forest@mt{east}%
5606 \forest@lt{north east}%
5607 \forest@lt{north}%
5608 \forest@lt{north west}%
5609 \forest@lt{west}%
5610 \forest@lt{south west}%
5611 \forest@lt{south}%
5612 \forest@lt{south east}%
5613 \forest@lt{east}%
5614 }%
5615 \csdef{forest@compute@node@boundary@chamfered rectangle}{%
5616 \forest@mt{before south west}%
5617 \forest@mt{after south west}%

```

```

5618 \forest@lt{before south east}%
5619 \forest@lt{after south east}%
5620 \forest@lt{before north east}%
5621 \forest@lt{after north east}%
5622 \forest@lt{before north west}%
5623 \forest@lt{after north west}%
5624 \forest@lt{before south west}%
5625 }%

```

7.3 Compute absolute positions

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

```

5626 \def\forest@node@computeabsolutepositions{%
5627   \edef\forest@marshal{%
5628     \noexpand\forest@node@foreachchild{%
5629       \noexpand\forest@node@computeabsolutepositions@{\forestove{x}}{\forestove{y}}{\forestove{grow}}}%
5630   }%
5631   }\forest@marshal
5632 }
5633 \def\forest@node@computeabsolutepositions@#1#2#3{%
5634   \pgfpointadd{\pgfpoint{#1}{#2}}{%
5635     \pgfpointadd{\pgfpolar{#3}{\forestove{l}}}{\pgfpolar{90 + #3}{\forestove{s}}}}}%
5636   \pgfgetlastxy\forest@temp@x\forest@temp@y
5637   \forestolet{x}\forest@temp@x
5638   \forestolet{y}\forest@temp@y
5639   \edef\forest@marshal{%
5640     \noexpand\forest@node@foreachchild{%
5641       \noexpand\forest@node@computeabsolutepositions@{\forest@temp@x}{\forest@temp@y}{\forestove{grow}}}%
5642   }%
5643   }\forest@marshal
5644 }

```

7.4 Drawing the tree

```

5645 \newif\ifforest@drawtree@preservenodeboxes@
5646 \def\forest@node@drawtree{%
5647   \expandafter\ifstrequal\expandafter{\forest@drawtreebox}{\pgfkeysnovalue}{%
5648     \let\forest@drawtree@beginbox\relax
5649     \let\forest@drawtree@endbox\relax
5650   }{%
5651     \edef\forest@drawtree@beginbox{\global\setbox\forest@drawtreebox=\hbox\bgroup}%
5652     \let\forest@drawtree@endbox\egroup
5653   }%
5654   \ifforest@external@
5655   \ifforest@externalize@tree@
5656   \forest@temptrue
5657   \else
5658     \tikzifexternalizing{%
5659       \ifforest@was@tikzexternalwasenable
5660       \forest@temptrue
5661       \pgfkeys{/tikz/external/optimize=false}%
5662       \let\forest@drawtree@beginbox\relax
5663       \let\forest@drawtree@endbox\relax
5664     \else
5665       \forest@tempfalse
5666     \fi
5667   }{%
5668     \forest@tempfalse
5669   }%

```

```

5670 \fi
5671 \ifforest@temp
5672 \advance\forest@externalize@inner@n 1
5673 \edef\forest@externalize@filename{%
5674 \tikzexternalrealjob-forest-\forest@externalize@outer@n
5675 \ifnum\forest@externalize@inner@n=0 \else.\the\forest@externalize@inner@n\fi}%
5676 \expandafter\tikzsetnextfilename\expandafter{\forest@externalize@filename}%
5677 \tikzexternalenable
5678 \pgfkeysalso{/tikz/external/remake next,/tikz/external/export next}%
5679 \fi
5680 \ifforest@externalize@tree@
5681 \typeout{forest: Invoking a recursive call to generate the external picture
5682 '\forest@externalize@filename' for the following context+code:
5683 '\expandafter\detokenize\expandafter{\forest@externalize@id}'}%
5684 \fi
5685 \fi
5686 %
5687 \ifforest\tikzcshack
5688 \let\forest@original@tikz@parse@node\tikz@parse@node
5689 \let\tikz@parse@node\forest@tikz@parse@node
5690 \fi
5691 \pgfkeysgetvalue{/forest/begin draw/.@cmd}\forest@temp@begindraw
5692 \pgfkeysgetvalue{/forest/end draw/.@cmd}\forest@temp@enddraw
5693 \edef\forest@marshal{%
5694 \noexpand\forest@drawtree@beginbox
5695 \expandonce{\forest@temp@begindraw\pgfkeysnovalue\pgfeov}%
5696 \noexpand\forest@node@drawtree@
5697 \expandonce{\forest@temp@enddraw\pgfkeysnovalue\pgfeov}%
5698 \noexpand\forest@drawtree@endbox
5699 }\forest@marshal
5700 \ifforest\tikzcshack
5701 \let\tikz@parse@node\forest@original@tikz@parse@node
5702 \fi
5703 %
5704 \ifforest@external@
5705 \ifforest@externalize@tree@
5706 \tikzexternaldisable
5707 \eappto\forest@externalize@checkimages{%
5708 \noexpand\forest@includeexternal@check{\forest@externalize@filename}%
5709 }%
5710 \expandafter\ifstrequal\expandafter{\forest@drawtreebox}{\pgfkeysnovalue}{%
5711 \eappto\forest@externalize@loadimages{%
5712 \noexpand\forest@includeexternal{\forest@externalize@filename}%
5713 }%
5714 }{%
5715 \eappto\forest@externalize@loadimages{%
5716 \noexpand\forest@includeexternal@box\forest@drawtreebox{\forest@externalize@filename}%
5717 }%
5718 }%
5719 \fi
5720 \fi
5721 }
5722 \def\forest@node@drawtree@{%
5723 \forest@forthis{\forestset{draw tree method}}%
5724 \forest@node@ifnamedefined{forest@baseline@node}{%
5725 \edef\forest@temp{%
5726 \noexpand\pgfsetbaselinepointlater{%
5727 \noexpand\pgfpntanchor
5728 {\forestOve{\forest@node@Nametoid{\forest@baseline@node}}{name}}
5729 {\forestOve{\forest@node@Nametoid{\forest@baseline@node}}{anchor}}
5730 }%

```

```

5731     }\forest@temp
5732 }{}%
5733 }
5734 \def\forest@draw@node{%
5735   \ifnum\forest@ve{phantom}=0
5736     \forest@node@forest@position@node@later@restore
5737     \ifforest@drawtree@preservenodeboxes@
5738       \pgfnodealias{forest@temp}{\forest@ve{later@name}}%
5739       \fi
5740       \pgfpositionnodenow{\pgfqpoint{\forest@ve{x}}{\forest@ve{y}}}%
5741       \ifforest@drawtree@preservenodeboxes@
5742       \pgfnodealias{\forest@ve{later@name}}{forest@temp}%
5743       \fi
5744   \fi
5745 }
5746 \def\forest@draw@edge{%
5747   \ifnum\forest@cn=\forest@root\relax\else
5748     \ifnum\forest@ve{phantom}=0
5749       \ifnum\forest@ve{\forest@ve{@parent}}{phantom}=0
5750         \edef\forest@temp{\forest@ve{edge path}}%
5751         \forest@temp
5752       \fi
5753     \fi
5754   \fi
5755 }
5756 \def\forest@draw@tikz{%
5757   \forest@ve{tikz}%
5758 }

```

8 Geometry

A *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.

```

5759 \newdimen\forest@xg
5760 \newdimen\forest@yg
5761 \newdimen\forest@xs
5762 \newdimen\forest@ys
5763 \def\forest@setupgrowline#1{%
5764   \edef\forest@grow{#1}%
5765   \pgfpointpolar\forest@grow{1pt}%
5766   \forest@xg=\pgf@x
5767   \forest@yg=\pgf@y
5768   \forest@xs=-\pgf@y
5769   \forest@ys=\pgf@x
5770 }

```

8.1 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`.

```

5771 \def\forest@pgfpointprojectiontogrowline#1{%
5772   \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.

```

5773   \pgfutil@tempdima=\pgf@sys@tonumber{\pgf@x}\forest@xg%
5774   \advance\pgfutil@tempdima by\pgf@sys@tonumber{\pgf@y}\forest@yg%

```

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

```
5775 \global\pgf@x=\pgf@sys@tonumber{\pgfutil@tempdima}\forest@xg%
5776 \global\pgf@y=\pgf@sys@tonumber{\pgfutil@tempdima}\forest@yg%
5777 }}
```

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.

```
5778 \def\forest@distancetogrowline#1#2{%
5779   \pgf@process{#2}%
5780   #1=\pgf@sys@tonumber{\pgf@x}\forest@xs\relax
5781   \advance#1 by\pgf@sys@tonumber{\pgf@y}\forest@ys\relax
5782 }
```

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

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

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

```
5783 \let\forest@pp@n\relax
5784 \def\forest@projectpathtogrowline#1#2{%
5785   \edef\forest@pp@prefix{#2}%
5786   \forest@save@pgfsyssoftpath@tokendefs
5787   \let\pgfsyssoftpath@movetotoken\forest@projectpath@processpoint
5788   \let\pgfsyssoftpath@linetotoken\forest@projectpath@processpoint
5789   \c@pgf@counta=0
5790   #1%
5791   \csedef{#2n}{\the\c@pgf@counta}%
5792   \forest@restore@pgfsyssoftpath@tokendefs
5793 }
```

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

```
5794 \def\forest@projectpath@processpoint#1#2{%
5795   \pgfqpoint{#1}{#2}%
5796   \expandafter\edef\c@pgf@counta\forest@pp@prefix\the\c@pgf@counta x\endcsname{\the\pgf@x}%
5797   \expandafter\edef\c@pgf@counta\forest@pp@prefix\the\c@pgf@counta y\endcsname{\the\pgf@y}%
5798   \forest@pgfp@point@projection@t@ogrowline{}%
5799   \expandafter\edef\c@pgf@counta\forest@pp@prefix\the\c@pgf@counta xp\endcsname{\the\pgf@x}%
5800   \expandafter\edef\c@pgf@counta\forest@pp@prefix\the\c@pgf@counta yp\endcsname{\the\pgf@y}%
5801   \advance\c@pgf@counta 1\relax
5802 }
```

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

```
5803 \def\forest@sortprojections#1{%
5804   % todo: optimize in cases when we know that the array is actually a
5805   % merger of sorted arrays; when does this happen? in
5806   % distance_between_paths, and when merging the edges of the parent
5807   % and its children in a uniform growth tree
5808   \edef\forest@ppi@inputprefix{#1}%
5809   \c@pgf@counta=\c@pgf@counta#1n\endcsname\relax
5810   \advance\c@pgf@counta -1
5811   \forest@sort\forest@ppi@inputprefix\forest@ppi@inputprefix\let\forest@sort@ascending{0}{\the\c@pgf@counta}%
5812 }
```

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

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

```
5813 \def\forest@processprojectioninfo#1#2{%
5814   \edef\forest@ppi

```

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

```
5815   \c@pgf@counta=0
5816   \c@pgf@countb=-1
5817   \safeloop
5818   \ifnum\c@pgf@counta<\csname#1n\endcsname\relax
```

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

```
5819   \letcs\forest@xo{#1\the\c@pgf@counta xo}%
5820   \letcs\forest@yo{#1\the\c@pgf@counta yo}%
5821   \letcs\forest@xp{#1\the\c@pgf@counta xp}%
5822   \letcs\forest@yp{#1\the\c@pgf@counta yp}%
5823   \ifnum\c@pgf@countb<0
5824     \forest@equaltotolerancefalse
5825   \else
5826     \forest@equaltotolerance
5827     {\pgfpoint\forest@xp\forest@yp}%
5828     {\pgfpoint
5829      {\csname#2\the\c@pgf@countb x\endcsname}%
5830      {\csname#2\the\c@pgf@countb y\endcsname}%
5831     }%
5832   \fi
5833   \ifforest@equaltotolerance\else
```

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

```
5834     \advance\c@pgf@countb 1
5835     \cslet{#2\the\c@pgf@countb x}\forest@xp
5836     \cslet{#2\the\c@pgf@countb y}\forest@yp
5837     \csdef{#2\the\c@pgf@countb @n}{0}%
5838   \fi
```

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

```
5839   % todo: this is ugly!
5840   \ifdefined\forest@xo\ifx\forest@xo\relax\else
5841     \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.

```
5842     \forest@append@point@to@inner@array
5843     \forest@xo\forest@yo
5844     {#2\the\c@pgf@countb @}%
```

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

```
5845     \csedef{#2(\forest@xo,\forest@yo)}{\the\c@pgf@countb}%
5846   \fi\fi
5847 \fi\fi
```

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.

Clean-up the raw array item.

```
5848 \cslet{#1\the\c@pgf@counta xo}\relax
5849 \cslet{#1\the\c@pgf@counta yo}\relax
5850 \cslet{#1\the\c@pgf@counta xp}\relax
5851 \cslet{#1\the\c@pgf@counta yp}\relax
5852 \advance\c@pgf@counta 1
5853 \saferepeat
```

Clean up the raw array length.

```
5854 \cslet{#1n}\relax
```

Store the length of the outer result array.

```
5855 \advance\c@pgf@countb 1
5856 \csedef{#2n}{\the\c@pgf@countb}%
5857 }
```

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

```
5858 \def\forest@ppiraw@let#1#2{%
5859 \csletcs{\forest@ppi@inputprefix#1xo}{\forest@ppi@inputprefix#2xo}%
5860 \csletcs{\forest@ppi@inputprefix#1yo}{\forest@ppi@inputprefix#2yo}%
5861 \csletcs{\forest@ppi@inputprefix#1xp}{\forest@ppi@inputprefix#2xp}%
5862 \csletcs{\forest@ppi@inputprefix#1yp}{\forest@ppi@inputprefix#2yp}%
5863 }
```

Item comparison macro for quicksorting the raw projection data.

```
5864 \def\forest@ppiraw@cmp#1#2{%
5865 \forest@sort@cmptwodimcs
5866 {\forest@ppi@inputprefix#1xp}{\forest@ppi@inputprefix#1yp}%
5867 {\forest@ppi@inputprefix#2xp}{\forest@ppi@inputprefix#2yp}%
5868 }
```

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

```
5869 \def\forest@append@point@to@inner@array#1#2#3{%
5870 \c@pgf@countc=\csname#3n\endcsname\relax
5871 \csedef{#3\the\c@pgf@countc x}{#1}%
5872 \csedef{#3\the\c@pgf@countc y}{#2}%
5873 \forest@distancetogrowline\pgfutil@tempdima{\pgfqpoint#1#2}%
5874 \csedef{#3\the\c@pgf@countc d}{\the\pgfutil@tempdima}%
5875 \advance\c@pgf@countc 1
5876 \csedef{#3n}{\the\c@pgf@countc}%
5877 }
```

8.2 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.

```
5878 \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.

```
5879 \edef\forest@bp@prefix{#2}%
5880 \forest@save@pgfsyssoftpath@tokendefs
5881 \let\pgfsyssoftpath@movetotoken\forest@breakpath@processfirstpoint
5882 \let\pgfsyssoftpath@linetotoken\forest@breakpath@processfirstpoint
5883 %\pgfusepath{}% empty the current path. ok?
5884 #1%
```

```

5885 \forest@restore@pgfsyssoftpath@tokendefs
5886 \pgfsyssoftpath@getcurrentpath#3%
5887 }

```

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.)

```

5888 \def\forest@breakpath@processfirstpoint#1#2{%
5889 \forest@breakpath@processmoveto{#1}{#2}%
5890 \let\pgfsyssoftpath@movetotoken\forest@breakpath@processmoveto
5891 \let\pgfsyssoftpath@linetotoken\forest@breakpath@processlineto
5892 }

```

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.

```

5893 \def\forest@breakpath@processmoveto#1#2{%
5894 \pgfsyssoftpath@moveto{#1}{#2}%
5895 \def\forest@previous@x{#1}%
5896 \def\forest@previous@y{#2}%
5897 \expandafter\let\expandafter\forest@previous@i
5898 \csname\forest@bp@prefix(#1,#2)\endcsname
5899 \expandafter\let\expandafter\forest@previous@px
5900 \csname\forest@bp@prefix\forest@previous@i x\endcsname
5901 \expandafter\let\expandafter\forest@previous@py
5902 \csname\forest@bp@prefix\forest@previous@i y\endcsname
5903 }

```

This is the heart of the path-breaking procedure.

```

5904 \def\forest@breakpath@processlineto#1#2{%

```

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

```

5905 \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.)

```

5906 \expandafter\let\expandafter\forest@i
5907 \csname\forest@bp@prefix(#1,#2)\endcsname
5908 \expandafter\let\expandafter\forest@px
5909 \csname\forest@bp@prefix\forest@i x\endcsname
5910 \expandafter\let\expandafter\forest@py
5911 \csname\forest@bp@prefix\forest@i y\endcsname

```

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

```

5912 \forest@equaltotolerance
5913 {\pgfqpoint{\forest@previous@px}{\forest@previous@py}}%
5914 {\pgfqpoint{\forest@px}{\forest@py}}%
5915 \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.

```

5916 \let\forest@breakpath@op\pgfsyssoftpath@moveto
5917 \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.

```

5918 \forest@temp@count=\forest@previous@i\relax
5919 \ifnum\forest@previous@i<\forest@i\relax
5920 \def\forest@breakpath@step{1}%
5921 \def\forest@breakpath@test{\forest@temp@count<\forest@i\relax}%
5922 \else
5923 \def\forest@breakpath@step{-1}%
5924 \def\forest@breakpath@test{\forest@temp@count>\forest@i\relax}%
5925 \fi

```

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

```
5926 \safeloop
5927 \advance\forest@temp@count\forest@breakpath@step\relax
5928 \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.)

```
5929 \pgfpointintersectionoflines
5930 {\pgfqpoint
5931 {\csname\forest@bp@prefix\the\forest@temp@count x\endcsname}%
5932 {\csname\forest@bp@prefix\the\forest@temp@count y\endcsname}%
5933 }%
5934 {\pgfpointadd
5935 {\pgfqpoint
5936 {\csname\forest@bp@prefix\the\forest@temp@count x\endcsname}%
5937 {\csname\forest@bp@prefix\the\forest@temp@count y\endcsname}%
5938 }%
5939 {\pgfqpoint{\forest@xs}{\forest@ys}}%
5940 }%
5941 {\pgfqpoint{\forest@previous@x}{\forest@previous@y}}%
5942 {\pgfqpoint{#1}{#2}}%
```

Break the segment at the intersection.

```
5943 \pgfgetlastxy\forest@last@x\forest@last@y
5944 \pgfsyssoftpath@lineto\forest@last@x\forest@last@y
```

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

```
5945 \forest@append@point@to@inner@array
5946 \forest@last@x\forest@last@y
5947 {\forest@bp@prefix\the\forest@temp@count @}%
```

Cache the projection of the new segment edge.

```
5948 \csedef{\forest@bp@prefix(\the\pgf@x,\the\pgf@y)}{\the\forest@temp@count}%
5949 \saferepeat
5950 \fi
```

Add the current point.

```
5951 \forest@breakpath@op{#1}{#2}%
```

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

```
5952 \def\forest@previous@x{#1}%
5953 \def\forest@previous@y{#2}%
5954 \let\forest@previous@i\forest@i
5955 \let\forest@previous@px\forest@px
5956 \let\forest@previous@py\forest@py
5957 }
```

8.3 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.

⁴For the definition of negative/positive side, see `forest@distancetogrowline` in §8.1

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.

```

5958 \def\forest@getnegativetightedgeofpath#1#2{%
5959   \forest@get@onetightedgeofpath#1\forest@sort@ascending#2}
5960 \def\forest@getpositivetightedgeofpath#1#2{%
5961   \forest@get@onetightedgeofpath#1\forest@sort@descending#2}
5962 \def\forest@get@onetightedgeofpath#1#2#3{%
5963   {%
5964     \forest@get@one@tightedgeofpath#1#2\forest@gep@edge
5965     \global\let\forest@gep@global@edge\forest@gep@edge
5966   }%
5967   \let#3\forest@gep@global@edge
5968 }
5969 \def\forest@get@one@tightedgeofpath#1#2#3{%

```

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

```

5970   \forest@projectpathtogrowline#1{forest@pp@}%
5971   \forest@sortprojections{forest@pp@}%
5972   \forest@processprojectioninfo{forest@pp@}{forest@pi@}%

```

Break the path.

```

5973   \forest@breakpath#1{forest@pi@}\forest@brokenpath

```

Compile some more useful information.

```

5974   \forest@sort@inner@arrays{forest@pi@}#2%
5975   \forest@pathtodict\forest@brokenpath{forest@pi@}%

```

The auxiliary data is set up: do the work!

```

5976   \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).

```

5977   \forest@simplifypath\forest@edge#3%
5978 }

```

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

```

5979 \def\forest@getbothtightedgesofpath#1#2#3{%
5980   {%
5981     \forest@get@one@tightedgeofpath#1\forest@sort@ascending\forest@gep@firstedge

```

Reverse the order of items in the inner arrays.

```

5982     \c@pgf@counta=0
5983     \forest@loop
5984     \ifnum\c@pgf@counta<\forest@pi@n\relax
5985       \forest@ppi@deflet{forest@pi@\the\c@pgf@counta @}%
5986       \forest@reversearray\forest@ppi@let
5987       {0}%
5988       {\csname forest@pi@\the\c@pgf@counta @n\endcsname}%
5989       \advance\c@pgf@counta 1
5990     \forest@repeat

```

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

```

5991     \forest@gettightedgeofpath@getedge\forest@edge
5992     \forest@simplifypath\forest@edge\forest@gep@secondedge

```

Smuggle the results out of the enclosing \TeX group.

```

5993     \global\let\forest@gep@global@firstedge\forest@gep@firstedge
5994     \global\let\forest@gep@global@secondedge\forest@gep@secondedge
5995   }%
5996   \let#2\forest@gep@global@firstedge
5997   \let#3\forest@gep@global@secondedge
5998 }

```

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

```

5999 \def\forest@sort@inner@arrays#1#2{%
6000   \c@pgf@counta=0
6001   \safeloop
6002   \ifnum\c@pgf@counta<\csname#1n\endcsname
6003     \c@pgf@countb=\csname#1\the\c@pgf@counta @n\endcsname\relax
6004     \ifnum\c@pgf@countb>1
6005       \advance\c@pgf@countb -1
6006       \forest@ppi@deflet{#1\the\c@pgf@counta @}%
6007       \forest@ppi@defcmp{#1\the\c@pgf@counta @}%
6008       \forest@sort\forest@ppi@cmp\forest@ppi@let#2{0}{\the\c@pgf@countb}%
6009     \fi
6010     \advance\c@pgf@counta 1
6011   \saferepeat
6012 }

```

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.

```

6013 \def\forest@ppi@deflet#1{%
6014   \edef\forest@ppi@let##1##2{%
6015     \noexpand\csletcs{#1##1x}{#1##2x}%
6016     \noexpand\csletcs{#1##1y}{#1##2y}%
6017     \noexpand\csletcs{#1##1d}{#1##2d}%
6018   }%
6019 }

```

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.

```

6020 \def\forest@ppi@defcmp#1{%
6021   \edef\forest@ppi@cmp##1##2{%
6022     \noexpand\forest@sort@cmpdimcs{#1##1d}{#1##2d}%
6023   }%
6024 }

```

Put path segments into a “segment dictionary”: for each segment of the path from (x_1, y_1) to (x_2, y_2)

let \forest@ (x_1, y_1) -- (x_2, y_2) be \forest@inpath (which can be anything but \relax).

```

6025 \let\forest@inpath\advance

```

This macro is just a wrapper to process the path.

```

6026 \def\forest@pathtodict#1#2{%
6027   \edef\forest@pathtodict@prefix{#2}%
6028   \forest@savepgfsyssoftpath@tokendefs
6029   \let\pgfsyssoftpath@movetotoken\forest@pathtodict@movetooop
6030   \let\pgfsyssoftpath@linetotoken\forest@pathtodict@linetooop
6031   \def\forest@pathtodict@subpathstart{ }%
6032   #1%
6033   \forest@restorepgfsyssoftpath@tokendefs
6034 }

```

When a move-to operation is encountered:

```

6035 \def\forest@pathtodict@movetooop#1#2{%

```

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

```

6036   \def\forest@pathtodict@subpathstart{(#1,#2)-}%
6037 }

```

When a line-to operation is encountered:

```

6038 \def\forest@pathtodict@linetooop#1#2{%

```

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

```

6039 \if\relax\forest@pathtodict@subpathstart\relax\else
6040   \let\forest@pathtodict@from\forest@pathtodict@subpathstart
6041   \fi

```

Mark the segment as existing.

```
6042 \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.

```
6043 \def\forest@pathtodict@from{(#1,#2)-}%
6044 \def\forest@pathtodict@subpathstart{}%
6045 }
```

In this macro, the edge is actually computed.

```
6046 \def\forest@gettightedgeofpath@getedge#1{% cs to store the edge into
```

Clear the path and the last projection.

```
6047 \pgfsyssoftpath@setcurrentpath\pgfutil@empty
6048 \let\forest@lastx\relax
6049 \let\forest@lasty\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.)

```
6050 \c@pgf@counta=0
6051 \forest@temp@count=\forest@pi@n\relax
6052 \advance\forest@temp@count -1
6053 \edef\forest@nminusone{\the\forest@temp@count}%
6054 \safeloop
6055 \ifnum\c@pgf@counta<\forest@nminusone\relax
6056 \forest@gettightedgeofpath@getedge@loopa
6057 \saferepeat
```

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

```
6058 \ifnum\forest@nminusone<\forest@n\relax\else
6059 \ifnum\csname forest@pi@\forest@nminusone @n\endcsname>0
6060 \forest@gettightedgeofpath@maybemoveto{\forest@nminusone}{0}%
6061 \fi
6062 \fi
6063 \pgfsyssoftpath@getcurrentpath#1%
6064 \pgfsyssoftpath@setcurrentpath\pgfutil@empty
6065 }
```

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.

```
6066 \def\forest@gettightedgeofpath@getedge@loopa{%
6067 \ifnum\csname forest@pi@\the\c@pgf@counta @n\endcsname>0
```

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

```
6068 \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.

```
6069 \c@pgf@countb=0
6070 \safeloop
6071 \ifnum\c@pgf@countb<\csname forest@pi@\the\c@pgf@counta @n\endcsname\relax
6072 \forest@gettightedgeofpath@getedge@loopb
6073 \saferepeat
6074 \fi
6075 \advance\c@pgf@counta 1
6076 }
```

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

```
6077 \def\forest@gettightedgeofpath@getedge@loopb{%
6078 \c@pgf@countc=0
6079 \advance\c@pgf@counta 1
```

```

6080 \edef\forest@aplusone{\the\c@pgf@counta}%
6081 \advance\c@pgf@counta -1
6082 \safeloop
6083 \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!

```

6084 \forest@tempfalse
6085 \expandafter\ifx\csname forest@pi@(%
6086 \csname forest@pi@\the\c@pgf@counta @\the\c@pgf@countb x\endcsname,%
6087 \csname forest@pi@\the\c@pgf@counta @\the\c@pgf@countb y\endcsname)--(%
6088 \csname forest@pi@\forest@aplusone @\the\c@pgf@countc x\endcsname,%
6089 \csname forest@pi@\forest@aplusone @\the\c@pgf@countc y\endcsname)%
6090 \endcsname\forest@inpath
6091 \forest@temptrue
6092 \else
6093 \expandafter\ifx\csname forest@pi@(%
6094 \csname forest@pi@\forest@aplusone @\the\c@pgf@countc x\endcsname,%
6095 \csname forest@pi@\forest@aplusone @\the\c@pgf@countc y\endcsname)--(%
6096 \csname forest@pi@\the\c@pgf@counta @\the\c@pgf@countb x\endcsname,%
6097 \csname forest@pi@\the\c@pgf@counta @\the\c@pgf@countb y\endcsname)%
6098 \endcsname\forest@inpath
6099 \forest@temptrue
6100 \fi
6101 \fi
6102 \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.

```

6103 \forest@gettightedgeofpath@maybemoveto{\the\c@pgf@counta}{\the\c@pgf@countb}%

```

Second, create a line to the end point.

```

6104 \edef\forest@last@x{%
6105 \csname forest@pi@\forest@aplusone @\the\c@pgf@countc x\endcsname}%
6106 \edef\forest@last@y{%
6107 \csname forest@pi@\forest@aplusone @\the\c@pgf@countc y\endcsname}%
6108 \pgfsyssoftpath@lineto\forest@last@x\forest@last@y

```

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

```

6109 \c@pgf@countc=\csname forest@pi@\forest@aplusone @n\endcsname
6110 \c@pgf@countb=\csname forest@pi@\the\c@pgf@counta @n\endcsname
6111 \fi
6112 \advance\c@pgf@countc 1
6113 \saferepeat
6114 \advance\c@pgf@countb 1
6115 }

```

`\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.

```

6116 \def\forest@gettightedgeofpath@maybemoveto#1#2{%

```



```

6117 \forest@temptrue
6118 \ifx\forest@last@x\relax\else
6119   \ifdim\forest@last@x=\csname forest@pi@#1@#2x\endcsname\relax
6120     \ifdim\forest@last@y=\csname forest@pi@#1@#2y\endcsname\relax
6121       \forest@tempfalse
6122     \fi
6123   \fi
6124 \fi
6125 \ifforest@temp
6126   \edef\forest@last@x{\csname forest@pi@#1@#2x\endcsname}%
6127   \edef\forest@last@y{\csname forest@pi@#1@#2y\endcsname}%
6128   \pgfsyssoftpath@moveto\forest@last@x\forest@last@y
6129 \fi
6130 }

```

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

```

6131 \def\forest@simplifypath#1#2{%
6132   \pgfsyssoftpath@setcurrentpath\pgfutil@empty
6133   \forest@save\pgfsyssoftpath@tokendefs
6134   \let\pgfsyssoftpath@movetotoken\forest@simplifypath@moveto
6135   \let\pgfsyssoftpath@linetotoken\forest@simplifypath@lineto
6136   \let\forest@last@x\relax
6137   \let\forest@last@y\relax
6138   \let\forest@last@atan\relax
6139   #1%
6140   \ifx\forest@last@x\relax\else
6141     \ifx\forest@last@atan\relax\else
6142       \pgfsyssoftpath@lineto\forest@last@x\forest@last@y
6143     \fi
6144   \fi
6145   \forest@restore\pgfsyssoftpath@tokendefs
6146   \pgfsyssoftpath@getcurrentpath#2%
6147   \pgfsyssoftpath@setcurrentpath\pgfutil@empty
6148 }

```

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.

```

6149 \def\forest@simplifypath@moveto#1#2{%
6150   \ifx\forest@last@x\relax\else
6151     \pgfsyssoftpath@lineto\forest@last@x\forest@last@y
6152   \fi
6153   \pgfsyssoftpath@moveto{#1}{#2}%
6154   \def\forest@last@x{#1}%
6155   \def\forest@last@y{#2}%
6156   \let\forest@last@atan\relax
6157 }

```

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

```

6158 \def\forest@getedgeofpath@precision{1pt}

```

When a line-to is encountered...

```

6159 \def\forest@simplifypath@lineto#1#2{%
6160   \ifx\forest@last@x\relax

```

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

```

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

```

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

```

6165 \pgfpointdiff{\pgfqpoint{#1}{#2}}{\pgfqpoint{\forest@last@x}{\forest@last@y}}%
6166 \ifdim\pgf@x<\pgfintersectiontolerance
6167 \ifdim-\pgf@x<\pgfintersectiontolerance
6168 \pgf@x=0pt
6169 \fi
6170 \fi
6171 \csname pgfmathatan2\endcsname{\pgf@x}{\pgf@y}%
6172 \let\forest@current@atan\pgfmathresult
6173 \ifx\forest@last@atan\relax

```

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

```

6174 \def\forest@last@x{#1}%
6175 \def\forest@last@y{#2}%
6176 \let\forest@last@atan\forest@current@atan
6177 \else

```

Otherwise, compare the first and the current slope.

```

6178 \pgfutil@tempdima=\forest@current@atan pt
6179 \advance\pgfutil@tempdima -\forest@last@atan pt
6180 \ifdim\pgfutil@tempdima<0pt\relax
6181 \multiply\pgfutil@tempdima -1
6182 \fi
6183 \ifdim\pgfutil@tempdima<\forest@getedgeofpath@precision\relax
6184 \else

```

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

```

6185 \pgfsyssoftpath@lineto\forest@last@x\forest@last@y
6186 \let\forest@last@atan\forest@current@atan
6187 \fi

```

In any event, update the last point.

```

6188 \def\forest@last@x{#1}%
6189 \def\forest@last@y{#2}%
6190 \fi
6191 \fi
6192 }

```

8.4 Get rectangle/band edge

```

6193 \def\forest@getnegativerectangleedgeofpath#1#2{%
6194 \forest@getnegativerectangleorbandedgeofpath{#1}{#2}{\the\pgf@xb}}
6195 \def\forest@getpositiverectangleedgeofpath#1#2{%
6196 \forest@getpositiverectangleorbandedgeofpath{#1}{#2}{\the\pgf@xb}}
6197 \def\forest@getbothrectangleedgesofpath#1#2#3{%
6198 \forest@getbothrectangleorbandedgesofpath{#1}{#2}{#3}{\the\pgf@xb}}
6199 \def\forest@bandlength{5000pt} % something large (ca. 180cm), but still manageable for TeX without producing
6200 \def\forest@getnegativebandedgeofpath#1#2{%
6201 \forest@getnegativerectangleorbandedgeofpath{#1}{#2}{\forest@bandlength}}
6202 \def\forest@getpositivebandedgeofpath#1#2{%
6203 \forest@getpositiverectangleorbandedgeofpath{#1}{#2}{\forest@bandlength}}
6204 \def\forest@getbothbandedgesofpath#1#2#3{%
6205 \forest@getbothrectangleorbandedgesofpath{#1}{#2}{#3}{\forest@bandlength}}
6206 \def\forest@getnegativerectangleorbandedgeofpath#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@ya}%
6210 \noexpand\pgfsyssoftpath@linetotoken{#3}{\the\pgf@ya}%
6211 }%
6212 {%
6213 \pgftransformreset

```

```

6214 \pgftransformrotate{\forest@grow}%
6215 \forest@pgfpathtransformed\forest@gre@path
6216 }%
6217 \pgfsyssoftpath@getcurrentpath#2%
6218 }
6219 \def\forest@getpositiverectangleorbandededgeofpath#1#2#3{%
6220 \forest@path@getboundingrectangle@ls#1{\forest@grow}%
6221 \edef\forest@gre@path{%
6222 \noexpand\pgfsyssoftpath@movetotoken{\the\pgf@xa}{\the\pgf@yb}%
6223 \noexpand\pgfsyssoftpath@linetotoken{#3}{\the\pgf@yb}%
6224 }%
6225 {%
6226 \pgftransformreset
6227 \pgftransformrotate{\forest@grow}%
6228 \forest@pgfpathtransformed\forest@gre@path
6229 }%
6230 \pgfsyssoftpath@getcurrentpath#2%
6231 }
6232 \def\forest@getbothrectangleorbandededgesofpath#1#2#3#4{%
6233 \forest@path@getboundingrectangle@ls#1{\forest@grow}%
6234 \edef\forest@gre@negpath{%
6235 \noexpand\pgfsyssoftpath@movetotoken{\the\pgf@xa}{\the\pgf@ya}%
6236 \noexpand\pgfsyssoftpath@linetotoken{#4}{\the\pgf@ya}%
6237 }%
6238 \edef\forest@gre@pospath{%
6239 \noexpand\pgfsyssoftpath@movetotoken{\the\pgf@xa}{\the\pgf@yb}%
6240 \noexpand\pgfsyssoftpath@linetotoken{#4}{\the\pgf@yb}%
6241 }%
6242 {%
6243 \pgftransformreset
6244 \pgftransformrotate{\forest@grow}%
6245 \forest@pgfpathtransformed\forest@gre@negpath
6246 }%
6247 \pgfsyssoftpath@getcurrentpath#2%
6248 {%
6249 \pgftransformreset
6250 \pgftransformrotate{\forest@grow}%
6251 \forest@pgfpathtransformed\forest@gre@pospath
6252 }%
6253 \pgfsyssoftpath@getcurrentpath#3%
6254 }

```

8.5 Distance between paths

Another crucial part of the package.

```

6255 \def\forest@distance@between@edge@paths#1#2#3{%
6256 % #1, #2 = (edge) paths
6257 %
6258 % project paths
6259 \forest@projectpathtogrowline#1{\forest@p1}%
6260 \forest@projectpathtogrowline#2{\forest@p2}%
6261 % merge projections (the lists are sorted already, because edge
6262 % paths are |sorted|)
6263 \forest@dbep@mergeprojections
6264 {\forest@p1}{\forest@p2}%
6265 {\forest@P1}{\forest@P2}%
6266 % process projections
6267 \forest@processprojectioninfo{\forest@P1}{\forest@PI1}%
6268 \forest@processprojectioninfo{\forest@P2}{\forest@PI2}%
6269 % break paths
6270 \forest@breakpath#1{\forest@PI1}\forest@broken@one

```

```

6271 \forest@breakpath#2{forest@PI2@}\forest@broken@two
6272 % sort inner arrays ---optimize: it's enough to find max and min
6273 \forest@sort@inner@arrays{forest@PI1@}\forest@sort@descending
6274 \forest@sort@inner@arrays{forest@PI2@}\forest@sort@ascending
6275 % compute the distance
6276 \let\forest@distance\relax
6277 \c@pgf@countc=0
6278 \forest@loop
6279 \ifnum\c@pgf@countc<\csname forest@PI1@n\endcsname\relax
6280 \ifnum\csname forest@PI1@\the\c@pgf@countc @n\endcsname=0 \else
6281 \ifnum\csname forest@PI2@\the\c@pgf@countc @n\endcsname=0 \else
6282 \pgfutil@tempdima=\csname forest@PI2@\the\c@pgf@countc @0d\endcsname\relax
6283 \advance\pgfutil@tempdima -\csname forest@PI1@\the\c@pgf@countc @0d\endcsname\relax
6284 \ifx\forest@distance\relax
6285 \edef\forest@distance{\the\pgfutil@tempdima}%
6286 \else
6287 \ifdim\pgfutil@tempdima<\forest@distance\relax
6288 \edef\forest@distance{\the\pgfutil@tempdima}%
6289 \fi
6290 \fi
6291 \fi
6292 \fi
6293 \advance\c@pgf@countc 1
6294 \forest@repeat
6295 \let#3\forest@distance
6296 }
6297 % merge projections: we need two projection arrays, both containing
6298 % projection points from both paths, but each with the original
6299 % points from only one path
6300 \def\forest@dbep@mergeprojections#1#2#3#4{%
6301 % TODO: optimize: v bistvu ni treba sortirat, ker je edge path e sortiran
6302 \forest@sortprojections{#1}%
6303 \forest@sortprojections{#2}%
6304 \c@pgf@counta=0
6305 \c@pgf@countb=0
6306 \c@pgf@countc=0
6307 \edef\forest@input@prefix@one{#1}%
6308 \edef\forest@input@prefix@two{#2}%
6309 \edef\forest@output@prefix@one{#3}%
6310 \edef\forest@output@prefix@two{#4}%
6311 \forest@dbep@mp@iterate
6312 \csedef{#3n}{\the\c@pgf@countc}%
6313 \csedef{#4n}{\the\c@pgf@countc}%
6314 }
6315 \def\forest@dbep@mp@iterate{%
6316 \let\forest@dbep@mp@next\forest@dbep@mp@iterate
6317 \ifnum\c@pgf@counta<\csname\forest@input@prefix@one n\endcsname\relax
6318 \ifnum\c@pgf@countb<\csname\forest@input@prefix@two n\endcsname\relax
6319 \let\forest@dbep@mp@next\forest@dbep@mp@do
6320 \else
6321 \let\forest@dbep@mp@next\forest@dbep@mp@iteratefirst
6322 \fi
6323 \else
6324 \ifnum\c@pgf@countb<\csname\forest@input@prefix@two n\endcsname\relax
6325 \let\forest@dbep@mp@next\forest@dbep@mp@iteratesecond
6326 \else
6327 \let\forest@dbep@mp@next\relax
6328 \fi
6329 \fi
6330 \forest@dbep@mp@next
6331 }

```

```

6332 \def\forest@dbep@mp@do{%
6333   \forest@sort@cmtwodimcs%
6334   {\forest@input@prefix@one\the\c@pgf@counta xp}%
6335   {\forest@input@prefix@one\the\c@pgf@counta yp}%
6336   {\forest@input@prefix@two\the\c@pgf@countb xp}%
6337   {\forest@input@prefix@two\the\c@pgf@countb yp}%
6338   \if\forest@sort@cmp@result=%
6339     \forest@dbep@mp@@store@p\forest@input@prefix@one\c@pgf@counta
6340     \forest@dbep@mp@@store@o\forest@input@prefix@one
6341     \c@pgf@counta\forest@output@prefix@one
6342     \forest@dbep@mp@@store@o\forest@input@prefix@two
6343     \c@pgf@countb\forest@output@prefix@two
6344     \advance\c@pgf@counta 1
6345     \advance\c@pgf@countb 1
6346   \else
6347     \if\forest@sort@cmp@result>%
6348       \forest@dbep@mp@@store@p\forest@input@prefix@two\c@pgf@countb
6349       \forest@dbep@mp@@store@o\forest@input@prefix@two
6350       \c@pgf@countb\forest@output@prefix@two
6351       \advance\c@pgf@countb 1
6352     \else%<
6353       \forest@dbep@mp@@store@p\forest@input@prefix@one\c@pgf@counta
6354       \forest@dbep@mp@@store@o\forest@input@prefix@one
6355       \c@pgf@counta\forest@output@prefix@one
6356       \advance\c@pgf@counta 1
6357     \fi
6358   \fi
6359   \advance\c@pgf@countc 1
6360   \forest@dbep@mp@iterate
6361 }
6362 \def\forest@dbep@mp@@store@p#1#2{%
6363   \csletcs
6364     {\forest@output@prefix@one\the\c@pgf@countc xp}%
6365     {#1\the#2xp}%
6366   \csletcs
6367     {\forest@output@prefix@one\the\c@pgf@countc yp}%
6368     {#1\the#2yp}%
6369   \csletcs
6370     {\forest@output@prefix@two\the\c@pgf@countc xp}%
6371     {#1\the#2xp}%
6372   \csletcs
6373     {\forest@output@prefix@two\the\c@pgf@countc yp}%
6374     {#1\the#2yp}%
6375 }
6376 \def\forest@dbep@mp@@store@o#1#2#3{%
6377   \csletcs{#3\the\c@pgf@countc xo}{#1\the#2xo}%
6378   \csletcs{#3\the\c@pgf@countc yo}{#1\the#2yo}%
6379 }
6380 \def\forest@dbep@mp@iteratefirst{%
6381   \forest@dbep@mp@iterateone\forest@input@prefix@one\c@pgf@counta\forest@output@prefix@one
6382 }
6383 \def\forest@dbep@mp@iteratesecond{%
6384   \forest@dbep@mp@iterateone\forest@input@prefix@two\c@pgf@countb\forest@output@prefix@two
6385 }
6386 \def\forest@dbep@mp@iterateone#1#2#3{%
6387   \forest@loop
6388   \ifnum#2<\csname#1n\endcsname\relax
6389     \forest@dbep@mp@@store@p#1#2%
6390     \forest@dbep@mp@@store@o#1#2#3%
6391     \advance\c@pgf@countc 1
6392     \advance#21

```

```

6393 \forest@repeat
6394 }

```

8.6 Utilities

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

```

6395 \newif\ifforest@equaltotolerance
6396 \def\forest@equaltotolerance#1#2{%
6397   \pgfpointdiff{#1}{#2}%
6398   \ifdim\pgf@x<0pt \multiply\pgf@x -1 \fi
6399   \ifdim\pgf@y<0pt \multiply\pgf@y -1 \fi
6400   \global\forest@equaltotolerancefalse
6401   \ifdim\pgf@x<\pgfintersectiontolerance\relax
6402     \ifdim\pgf@y<\pgfintersectiontolerance\relax
6403       \global\forest@equaltolerancetrue
6404     \fi
6405   \fi
6406 }}

```

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

```

6407 \def\forest@save@pgfsyssoftpath@tokendefs{%
6408   \let\forest@origmovetotoken\pgfsyssoftpath@movetotoken
6409   \let\forest@origlinetotoken\pgfsyssoftpath@linetotoken
6410   \let\forest@origcurvetosupportatoken\pgfsyssoftpath@curvetosupportatoken
6411   \let\forest@origcurvetosupportbtoken\pgfsyssoftpath@curvetosupportbtoken
6412   \let\forest@origcurvetotoken\pgfsyssoftpath@curvetototoken
6413   \let\forest@origrectcornertoken\pgfsyssoftpath@rectcornertoken
6414   \let\forest@origrectsizetoken\pgfsyssoftpath@rectsizetoken
6415   \let\forest@origclosepathtoken\pgfsyssoftpath@closepathtoken
6416   \let\pgfsyssoftpath@movetotoken\forest@badtoken
6417   \let\pgfsyssoftpath@linetotoken\forest@badtoken
6418   \let\pgfsyssoftpath@curvetosupportatoken\forest@badtoken
6419   \let\pgfsyssoftpath@curvetosupportbtoken\forest@badtoken
6420   \let\pgfsyssoftpath@curvetototoken\forest@badtoken
6421   \let\pgfsyssoftpath@rectcornertoken\forest@badtoken
6422   \let\pgfsyssoftpath@rectsizetoken\forest@badtoken
6423   \let\pgfsyssoftpath@closepathtoken\forest@badtoken
6424 }
6425 \def\forest@badtoken{%
6426   \PackageError{forest}{This token should not be in this path}{}%
6427 }
6428 \def\forest@restore@pgfsyssoftpath@tokendefs{%
6429   \let\pgfsyssoftpath@movetotoken\forest@origmovetotoken
6430   \let\pgfsyssoftpath@linetotoken\forest@origlinetotoken
6431   \let\pgfsyssoftpath@curvetosupportatoken\forest@origcurvetosupportatoken
6432   \let\pgfsyssoftpath@curvetosupportbtoken\forest@origcurvetosupportbtoken
6433   \let\pgfsyssoftpath@curvetototoken\forest@origcurvetotoken
6434   \let\pgfsyssoftpath@rectcornertoken\forest@origrectcornertoken
6435   \let\pgfsyssoftpath@rectsizetoken\forest@origrectsizetoken
6436   \let\pgfsyssoftpath@closepathtoken\forest@origclosepathtoken
6437 }

```

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

```

6438 \def\forest@extendpath#1#2#3{%
6439   \pgf@process{#3}%
6440   \pgfsyssoftpath@setcurrentpath#1%
6441   \forest@save@pgfsyssoftpath@tokendefs
6442   \let\pgfsyssoftpath@movetotoken\forest@extendpath@moveto
6443   \let\pgfsyssoftpath@linetotoken\forest@extendpath@lineto
6444   #2%

```

```

6445 \forest@restore@pgfsyssoftpath@tokendefs
6446 \pgfsyssoftpath@getcurrentpath#1%
6447 }
6448 \def\forest@extendpath@moveto#1#2{%
6449 \forest@extendpath@do{#1}{#2}\pgfsyssoftpath@moveto
6450 }
6451 \def\forest@extendpath@lineto#1#2{%
6452 \forest@extendpath@do{#1}{#2}\pgfsyssoftpath@lineto
6453 }
6454 \def\forest@extendpath@do#1#2#3{%
6455 {%
6456 \advance\pgf@x #1
6457 \advance\pgf@y #2
6458 #3{\the\pgf@x}{\the\pgf@y}%
6459 }%
6460 }

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.)
6461 \def\forest@path@getboundingrectangle@ls#1#2{%
6462 {%
6463 \pgftransformreset
6464 \pgftransformrotate{-(#2)}%
6465 \forest@pgfpathtransformed#1%
6466 }%
6467 \pgfsyssoftpath@getcurrentpath\forest@gbr@rotatedpath
6468 \forest@path@getboundingrectangle@xy\forest@gbr@rotatedpath
6469 }
6470 \def\forest@path@getboundingrectangle@xy#1{%
6471 \forest@save@pgfsyssoftpath@tokendefs
6472 \let\pgfsyssoftpath@movetotoken\forest@gbr@firstpoint
6473 \let\pgfsyssoftpath@linetotoken\forest@gbr@firstpoint
6474 #1%
6475 \forest@restore@pgfsyssoftpath@tokendefs
6476 }
6477 \def\forest@gbr@firstpoint#1#2{%
6478 \pgf@xa=#1 \pgf@xb=#1 \pgf@ya=#2 \pgf@yb=#2
6479 \let\pgfsyssoftpath@movetotoken\forest@gbr@point
6480 \let\pgfsyssoftpath@linetotoken\forest@gbr@point
6481 }
6482 \def\forest@gbr@point#1#2{%
6483 \ifdim#1<\pgf@xa\relax\pgf@xa=#1 \fi
6484 \ifdim#1>\pgf@xb\relax\pgf@xb=#1 \fi
6485 \ifdim#2<\pgf@ya\relax\pgf@ya=#2 \fi
6486 \ifdim#2>\pgf@yb\relax\pgf@yb=#2 \fi
6487 }

```

9 The outer UI

9.1 Externalization

```

6488 \pgfkeys{/forest/external/.cd,
6489 copy command/.initial={cp "\source" "\target"},
6490 optimize/.is if=forest@external@optimize@,
6491 context/.initial={%
6492 \forestOve{csname forest@id@of@standard node@endcsname}{environment@formula}},
6493 depends on macro/.style={context/.append/.expanded={%
6494 \expandafter\detokenize\expandafter{#1}}},
6495 }
6496 \def\forest@external@copy#1#2{%

```

```

6497 \pgfkeysgetvalue{/forest/external/copy command}\forest@copy@command
6498 \ifx\forest@copy@command\pgfkeysnovalue\else
6499 \IfFileExists{#1}{%
6500   {%
6501     \def\source{#1}%
6502     \def\target{#2}%
6503     \immediate\write18{\forest@copy@command}%
6504   }%
6505 }{%
6506 \fi
6507 }
6508 \newif\ifforest@external@optimize@
6509 \forest@external@optimize@true
6510 \ifforest@install@keys@to@tikz@path@
6511 \tikzset{
6512   fit to/.style={
6513     /forest/for nodewalk=%
6514     {TeX={\def\forest@fitto{}}},#1}%
6515     {TeX={\eappto\forest@fitto{(\forestove{name})}}},
6516   fit/.expanded={\forest@fitto}
6517 },
6518 }
6519 \fi
6520 \ifforest@external@
6521 \ifdefined\tikzexternal@tikz@replacement\else
6522 \usetikzlibrary{external}%
6523 \fi
6524 \pgfkeys{%
6525   /tikz/external/failed ref warnings for={},
6526   /pgf/images/aux in dpth=false,
6527 }%
6528 \tikzifexternalizing{}{%
6529   \forest@external@copy{\jobname.aux}{\jobname.aux.copy}%
6530 }%
6531 \AtBeginDocument{%
6532   \tikzifexternalizing{%
6533     \IfFileExists{\tikzexternalrealjob.aux.copy}{%
6534       \makeatletter
6535       \input \tikzexternalrealjob.aux.copy
6536       \makeatother
6537     }{%
6538     }{%
6539       \newwrite\forest@auxout
6540       \immediate\openout\forest@auxout=\tikzexternalrealjob.for.tmp
6541     }%
6542     \IfFileExists{\tikzexternalrealjob.for}{%
6543       {%
6544         \makehashother\makeatletter
6545         \input \tikzexternalrealjob.for
6546       }%
6547     }{%
6548     }%
6549   \AtEndDocument{%
6550     \tikzifexternalizing{}{%
6551       \immediate\closeout\forest@auxout
6552       \forest@external@copy{\jobname.for.tmp}{\jobname.for}%
6553     }%
6554   }%
6555 \fi

```


9.2 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.

```
6556 \NewDocumentEnvironment{forest}{D()}{stages}}{%
6557   \forest@defstages{#1}%
6558   \Collect@Body
6559   \forest@env
6560 }{}
6561 \NewDocumentCommand{\Forest}{s D()}{stages} m){%
6562   \forest@defstages{#2}%
6563   \IfBooleanTF{#1}{\let\forest@next\forest@env}{\let\forest@next\forest@group@env}%
6564   \forest@next{#3}%
6565 }
6566 \def\forest@defstages#1{%
6567   \def\forest@stages{#1}%
6568 }
6569 \def\forest@group@env#1{{\forest@env{#1}}}
6570 \newif\ifforest@externalize@tree@
6571 \newif\ifforest@was@tikzexternalwasenable
6572 \newcommand\forest@env[1]{%
6573   \let\forest@external@next\forest@begin
6574   \forest@was@tikzexternalwasenablefalse
6575   \ifdefined\tikzexternal@tikz@replacement
6576     \ifx\tikz\tikzexternal@tikz@replacement
6577       \forest@was@tikzexternalwasenabletrue
6578       \tikzexternaldisable
6579     \fi
6580   \fi
6581   \forest@externalize@tree@false
6582   \ifforest@external@
6583     \ifforest@was@tikzexternalwasenable
6584     \forest@env@
6585   \fi
6586   \fi
6587   \forest@standardnode@calibrate
6588   \forest@external@next{#1}%
6589 }
6590 \def\forest@env@{%
6591   \iftikzexternal@exportnext
6592     \tikzifexternalizing{%
6593       \let\forest@external@next\forest@begin@externalizing
6594     }{%
6595       \let\forest@external@next\forest@begin@externalize
6596     }%
6597   \else
6598     \tikzexternal@exportnexttrue
6599   \fi
6600 }
```

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

```
6601 \long\def\forest@begin@externalizing#1{%
6602   \forest@external@setup{#1}%
6603   \let\forest@external@next\forest@begin
6604   \forest@externalize@inner@n=-1
6605   \ifforest@external@optimize@\forest@externalizing@maybeoptimize\fi
6606   \forest@external@next{#1}%
6607   \tikzexternalenable
6608 }
6609 \def\forest@externalizing@maybeoptimize{%
```

```

6610 \edef\forest@temp{\tikzexternalrealjob-forest-\forest@externalize@outer@n}%
6611 \edef\forest@marshal{%
6612   \noexpand\pgfutil@in@
6613     {\expandafter\detokenize\expandafter{\forest@temp}.}
6614     {\expandafter\detokenize\expandafter{\pgfactualjobname}.}%
6615 } \forest@marshal
6616 \ifpgfutil@in@
6617 \else
6618   \let\forest@external@next\@gobble
6619 \fi
6620 }

```

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

```

6621 \long\def\forest@begin@externalize#1{%
6622   \forest@external@setup{#1}%
6623   \iftikzexternal@file@isuptodate
6624     \setbox0=\hbox{%
6625       \csname forest@externalcheck@\forest@externalize@outer@n\endcsname
6626     }%
6627   \fi
6628   \iftikzexternal@file@isuptodate
6629     \csname forest@externalload@\forest@externalize@outer@n\endcsname
6630   \else
6631     \forest@externalize@tree@true
6632     \forest@externalize@inner@n=-1
6633     \forest@begin{#1}%
6634     \ifcsdef{forest@externalize@@\forest@externalize@id}{-}{%
6635       \immediate\write\forest@auxout{%
6636         \noexpand\forest@external
6637         {\forest@externalize@outer@n}%
6638         {\expandafter\detokenize\expandafter{\forest@externalize@id}}%
6639         {\expandonce\forest@externalize@checkimages}%
6640         {\expandonce\forest@externalize@loadimages}%
6641       }%
6642     }%
6643   \fi
6644   \tikzexternalenable
6645 }
6646 \def\forest@includeexternal@check#1{%
6647   \tikzsetnextfilename{#1}%
6648   \IfFileExists{\tikzexternal@filenameprefix/#1}{\tikzexternal@file@isuptodate true}{\tikzexternal@file@isuptodate false}
6649 }
6650 \def\makehashother{\catcode'\# =12}%
6651 \long\def\forest@external@setup#1{%
6652   % set up \forest@externalize@id and \forest@externalize@outer@n
6653   % we need to deal with #s correctly (\write doubles them)
6654   \setbox0=\hbox{\makehashother\makeatletter
6655     \scantokens{\forest@temp@toks{#1}}\expandafter
6656   }%
6657   \expandafter\forest@temp@toks\expandafter{\the\forest@temp@toks}%
6658   \edef\forest@temp{\pgfkeysvalueof{/forest/external/context}}%
6659   \edef\forest@externalize@id{%
6660     \expandafter\detokenize\expandafter{\forest@temp}%
6661     @@%
6662     \expandafter\detokenize\expandafter{\the\forest@temp@toks}%
6663   }%
6664   \letcs\forest@externalize@outer@n{forest@externalize@@\forest@externalize@id}%
6665   \ifdefined\forest@externalize@outer@n
6666     \global\tikzexternal@file@isuptodate true
6667   \else
6668     \global\advance\forest@externalize@max@outer@n 1

```

```

6669 \edef\forest@externalize@outer@n{\the\forest@externalize@max@outer@n}%
6670 \global\tikzexternal@file@isuptodatefalse
6671 \fi
6672 \def\forest@externalize@loadimages{%
6673 \def\forest@externalize@checkimages{%
6674 }
6675 \newcount\forest@externalize@max@outer@n
6676 \global\forest@externalize@max@outer@n=0
6677 \newcount\forest@externalize@inner@n

```

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

```

6678 \long\def\forest@externalize#1#2#3#4{% #1=n,#2=context+source code,#3=update check code, #4=load code
6679 \ifnum\forest@externalize@max@outer@n<#1
6680 \global\forest@externalize@max@outer@n=#1
6681 \fi
6682 \global\csdef{forest@externalize@@\detokenize{#2}}{#1}%
6683 \global\csdef{forest@externalcheck@#1}{#3}%
6684 \global\csdef{forest@externalload@#1}{#4}%
6685 \tikzifexternalizing{}{%
6686 \immediate\write\forest@auxout{%
6687 \noexpand\forest@external{#1}%
6688 {\expandafter\detokenize\expandafter{#2}}%
6689 {\unexpanded{#3}}%
6690 {\unexpanded{#4}}%
6691 }%
6692 }%
6693 }

```

These two macros include the external picture.

```

6694 \def\forest@includeexternal#1{%
6695 \edef\forest@temp{\pgfkeysvalueof{/forest/external/context}}%
6696 %\typeout{forest: Including external picture '#1' for forest context+code: '\expandafter\detokenize\expand
6697 {%
6698 %\def\pgf@declaredraftimage##1##2{\def\pgf@image{\hbox{}}}%
6699 \tikzsetnextfilename{#1}%
6700 \tikzexternalenable
6701 \tikz{}%
6702 }%
6703 }
6704 \def\forest@includeexternal@box#1#2{%
6705 \global\setbox#1=\hbox{\forest@includeexternal{#2}}%
6706 }

```

This code runs the bracket parser and stage processing.

```

6707 \long\def\forest@begin#1{%
6708 \iffalse{\fi\forest@parsebracket#1}%
6709 }
6710 \def\forest@parsebracket{%
6711 \bracketParse{\forest@get@root@afterthought}\forest@root=%
6712 }
6713 \def\forest@get@root@afterthought{%
6714 \expandafter\forest@get@root@afterthought\expandafter{\iffalse}\fi
6715 }
6716 \long\def\forest@get@root@afterthought@#1{%
6717 \ifblank{#1}{-}{%
6718 \forest@eappto{\forest@root}{given options}{,afterthought={\unexpanded{#1}}}%
6719 }%
6720 \forest@do
6721 }
6722 \def\forest@do{%
6723 \forest@node@Compute@numeric@ts@info{\forest@root}%
6724 \expandafter\forestset\expandafter{\forest@stages}%

```

```

6725 \ifforest@was@tikzexternalwasenable
6726 \tikzexternalenable
6727 \fi
6728 }

```

9.3 Standard node

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

```

6729 \def\forest@standardnode@new{%
6730 \advance\forest@node@maxid1
6731 \forest@for@node{\the\forest@node@maxid}{%
6732 \forest@node@init
6733 \forest@node@setname{standard node}%
6734 }%
6735 }
6736 \def\forest@standardnode@calibrate{%
6737 \forest@for@node{\forest@node@Nametoid{standard node}}{%
6738 \edef\forest@environment{\forest@environment@formula}}%
6739 \forest@toget{previous@environment}\forest@previous@environment
6740 \ifx\forest@environment\forest@previous@environment\else
6741 \forest@tolet{previous@environment}\forest@environment
6742 \forest@node@typeset
6743 \forest@toget{calibration@procedure}\forest@temp
6744 \expandafter\forestset\expandafter{\forest@temp}%
6745 \fi
6746 }%
6747 }

```

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.)

```

6748 \def\forestStandardNode[#1]#2#3#4{%
6749 \let\forest@standardnode@restoretikzexternal\relax
6750 \ifdefined\tikzexternaldisable
6751 \ifx\tikz\tikzexternal\tikz@replacement
6752 \tikzexternaldisable
6753 \let\forest@standardnode@restoretikzexternal\tikzexternalenable
6754 \fi
6755 \fi
6756 \forest@standardnode@new
6757 \forest@for@node{\forest@node@Nametoid{standard node}}{%
6758 \forestset{content=#1}%
6759 \forest@set{environment@formula}{#2}%
6760 \edef\forest@temp{\unexpanded{#3}}%
6761 \forest@tolet{calibration@procedure}\forest@temp
6762 \def\forest@calibration@initializing@code{%
6763 \pgfqkeys{/forest/initializing@code}{#4}%
6764 \forest@tolet{initializing@code}\forest@calibration@initializing@code
6765 \forest@standardnode@restoretikzexternal
6766 }
6767 }
6768 \forestset{initializing@code/.unknown/.code={%
6769 \eappto\forest@calibration@initializing@code{%
6770 \noexpand\forest@get{\forest@node@Nametoid{standard node}}{\pgfkeyscurrentname}\noexpand\forest@temp
6771 \noexpand\forest@tolet{\pgfkeyscurrentname}\noexpand\forest@temp

```

```

6772 }%
6773 }
6774 }

```

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

```

6775 \def\forest@initializefromstandardnode{%
6776   \forest@ve{\forest@node@Nametoid{standard node}}{initializing@code}%
6777 }

```

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

```

6778 \forestStandardNode[dj]
6779 {%
6780   \forest@ve{\forest@node@Nametoid{standard node}}{content},%
6781   \the\ht\strutbox,\the\pgflinewidth,%
6782   \pgfkeysvalueof{/pgf/inner ysep},\pgfkeysvalueof{/pgf/outer ysep},%
6783   \pgfkeysvalueof{/pgf/inner xsep},\pgfkeysvalueof{/pgf/outer xsep}%
6784 }
6785 {
6786   l sep={\the\ht\strutbox+\pgfkeysvalueof{/pgf/inner ysep}},
6787   l={l sep()+abs(max_y()-min_y())+2*\pgfkeysvalueof{/pgf/outer ysep}},
6788   s sep={2*\pgfkeysvalueof{/pgf/inner xsep}}
6789 }
6790 {l sep,l,s sep}

```

9.4 ls coordinate system

```

6791 \pgfkeys{/forest/@cs}{%
6792   name/.code={%
6793     \edef\forest@cn{\forest@node@Nametoid{#1}}%
6794     \forest@forestcs@resetxy},
6795   id/.code={%
6796     \edef\forest@cn{#1}%
6797     \forest@forestcs@resetxy},
6798   go/.code={%
6799     \forest@go{#1}%
6800     \forest@forestcs@resetxy},
6801   anchor/.code={\forest@forestcs@anchor{#1}},
6802   l/.code={%
6803     \pgfmathsetlengthmacro\forest@forestcs@l{#1}%
6804     \forest@forestcs@ls
6805   },
6806   s/.code={%
6807     \pgfmathsetlengthmacro\forest@forestcs@s{#1}%
6808     \forest@forestcs@ls
6809   },
6810   .unknown/.code={%
6811     \expandafter\pgfutil@in@\expandafter{\pgfkeyscurrentname}%
6812     \ifpgfutil@in@
6813       \expandafter\forest@forestcs@namegoanchor\pgfkeyscurrentname\forest@end
6814     \else
6815       \expandafter\forest@nameandgo\expandafter{\pgfkeyscurrentname}%
6816       \forest@forestcs@resetxy
6817     \fi

```

```

6818 }
6819 }
6820 \def\forest@forestcs@resetxy{%
6821   \ifnum\forest@cn=0 \forest@cs@invalidnodeerror\fi
6822   \global\pgf@x\forestove{x}%
6823   \global\pgf@y\forestove{y}%
6824 }
6825 \def\forest@forestcs@ls{%
6826   \ifdefined\forest@forestcs@l
6827     \ifdefined\forest@forestcs@s
6828       {%
6829         \pgftransformreset
6830         \pgftransformrotate{\forestove{grow}}%
6831         \pgfpointransformed{\pgfpoint{\forest@forestcs@l}{\forest@forestcs@s}}%
6832       }%
6833       \global\advance\pgf@x\forestove{x}%
6834       \global\advance\pgf@y\forestove{y}%
6835     \fi
6836   \fi
6837 }
6838 \def\forest@forestcs@anchor#1{%
6839   \edef\forest@marshal{%
6840     \noexpand\forest@original@tikz@parse@node\relax
6841     (\forestove{name}\ifx\relax#1\relax\else.\fi#1)%
6842   }\forest@marshal
6843 }
6844 \def\forest@forestcs@namegoanchor#1.#2\forest@end{%
6845   \forest@nameandgo{#1}%
6846   \ifnum\forest@cn=0 \forest@cs@invalidnodeerror\fi
6847   \forest@forestcs@anchor{#2}%
6848 }
6849 \def\forest@cs@invalidnodeerror{%
6850   \PackageError{forest}{Attempt to refer to the invalid node by "forest cs"}%
6851 }
6852 \tikzdeclarecoordinatesystem{forest}{%
6853   \forest@forthis{%
6854     \forest@forestcs@resetxy
6855     \ifdefined\forest@forestcs@l\undef\forest@forestcs@l\fi
6856     \ifdefined\forest@forestcs@s\undef\forest@forestcs@s\fi
6857     \pgfqkeys{/forest/@cs}{#1}%
6858   }%
6859 }

```

9.5 Relative node names in TikZ

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

```

6860 \def\forest@tikz@parse@node#1(#2){%
6861   \pgfutil@in@.#2}%
6862   \ifpgfutil@in@
6863     \expandafter\forest@tikz@parse@node@checkiftikzname@withdot
6864   \else%
6865     \expandafter\forest@tikz@parse@node@checkiftikzname@withoutdot
6866   \fi%
6867   #1(#2)\forest@end
6868 }
6869 \def\forest@tikz@parse@node@checkiftikzname@withdot#1(#2.#3)\forest@end{%
6870   \forest@tikz@parse@node@checkiftikzname#1{#2}{.#3}}
6871 \def\forest@tikz@parse@node@checkiftikzname@withoutdot#1(#2)\forest@end{%
6872   \forest@tikz@parse@node@checkiftikzname#1{#2}{}}
6873 \def\forest@tikz@parse@node@checkiftikzname#1#2#3{%
6874   \expandafter\ifx\csname pgf@sh@ns@#2\endcsname\relax

```

```

6875 \forest@forthis{%
6876 \forest@nameandgo{#2}%
6877 \ifnum\forest@cn=0 \forest@cs@invalidnodeerror\fi
6878 \edef\forest@temp@relativenodename{\forest@ve{name}}%
6879 }%
6880 \else
6881 \def\forest@temp@relativenodename{#2}%
6882 \fi
6883 \expandafter\forest@original@tikz@parse@node\expandafter#1\expandafter(\forest@temp@relativenodename#3)%
6884 }
6885 \def\forest@nameandgo#1{%
6886 \pgfutil@in@!{#1}%
6887 \ifpgfutil@in@
6888 \forest@nameandgo@(#1)%
6889 \else
6890 \ifstrempy{#1}{}{\edef\forest@cn{\forest@node@Nametoid{#1}}}%
6891 \fi
6892 }
6893 \def\forest@nameandgo@(#1!#2){%
6894 \ifstrempy{#1}{}{\edef\forest@cn{\forest@node@Nametoid{#1}}}%
6895 \forest@go{#2}%
6896 }

```

9.6 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.

```

6897 \forestset{
6898 declare count={rotate}{0},
6899 autoforward'={rotate}{node options},
6900 }

```

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

```

6901 \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.

```

6902 \newif\ifforest@anchor@forwardtotikz

```

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

```

6903 \newif\ifforest@anchor@isborder

```

The UI macro.

```

6904 \def\forestanchortotikzanchor#1#2{% #1 = forest anchor, #2 = macro to receive the tikz anchor
6905 \forest@anchor@forwardtotikzfalse
6906 \forest@anchor@do{#1}{\forest@cn}%
6907 \let#2\forest@temp@anchor
6908 }

```

Generic anchors.

```

6909 \pgfdeclaregenericanchor{child anchor}{%
6910 \forest@anchor@forwardtotikztrue
6911 \forest@anchor@do{#1}{child anchor}{\forest@referencednodeid}%
6912 }
6913 \pgfdeclaregenericanchor{parent anchor}{%
6914 \forest@anchor@forwardtotikztrue
6915 \forest@anchor@do{#1}{parent anchor}{\forest@referencednodeid}%
6916 }
6917 \pgfdeclaregenericanchor{anchor}{%

```

```

6918 \forest@anchor@forwardtotikztrue
6919 \forest@anchor@do{#1}{anchor}{\forest@referencednodeid}%
6920 }
6921 \pgfdeclaregenericanchor{children}{%
6922 \forest@anchor@forwardtotikztrue
6923 \forest@anchor@do{#1}{children}{\forest@referencednodeid}%
6924 }
6925 \pgfdeclaregenericanchor{children first}{%
6926 \forest@anchor@forwardtotikztrue
6927 \forest@anchor@do{#1}{children first}{\forest@referencednodeid}%
6928 }
6929 \pgfdeclaregenericanchor{first}{%
6930 \forest@anchor@forwardtotikztrue
6931 \forest@anchor@do{#1}{first}{\forest@referencednodeid}%
6932 }
6933 \pgfdeclaregenericanchor{parent first}{%
6934 \forest@anchor@forwardtotikztrue
6935 \forest@anchor@do{#1}{parent first}{\forest@referencednodeid}%
6936 }
6937 \pgfdeclaregenericanchor{parent}{%
6938 \forest@anchor@forwardtotikztrue
6939 \forest@anchor@do{#1}{parent}{\forest@referencednodeid}%
6940 }
6941 \pgfdeclaregenericanchor{parent last}{%
6942 \forest@anchor@forwardtotikztrue
6943 \forest@anchor@do{#1}{parent last}{\forest@referencednodeid}%
6944 }
6945 \pgfdeclaregenericanchor{last}{%
6946 \forest@anchor@forwardtotikztrue
6947 \forest@anchor@do{#1}{last}{\forest@referencednodeid}%
6948 }
6949 \pgfdeclaregenericanchor{children last}{%
6950 \forest@anchor@forwardtotikztrue
6951 \forest@anchor@do{#1}{children last}{\forest@referencednodeid}%
6952 }
6953 \pgfdeclaregenericanchor{children'}{%
6954 \forest@anchor@forwardtotikztrue
6955 \forest@anchor@do{#1}{children'}{\forest@referencednodeid}%
6956 }
6957 \pgfdeclaregenericanchor{children first'}{%
6958 \forest@anchor@forwardtotikztrue
6959 \forest@anchor@do{#1}{children first'}{\forest@referencednodeid}%
6960 }
6961 \pgfdeclaregenericanchor{first'}{%
6962 \forest@anchor@forwardtotikztrue
6963 \forest@anchor@do{#1}{first'}{\forest@referencednodeid}%
6964 }
6965 \pgfdeclaregenericanchor{parent first'}{%
6966 \forest@anchor@forwardtotikztrue
6967 \forest@anchor@do{#1}{parent first'}{\forest@referencednodeid}%
6968 }
6969 \pgfdeclaregenericanchor{parent'}{%
6970 \forest@anchor@forwardtotikztrue
6971 \forest@anchor@do{#1}{parent'}{\forest@referencednodeid}%
6972 }
6973 \pgfdeclaregenericanchor{parent last'}{%
6974 \forest@anchor@forwardtotikztrue
6975 \forest@anchor@do{#1}{parent last'}{\forest@referencednodeid}%
6976 }
6977 \pgfdeclaregenericanchor{last'}{%
6978 \forest@anchor@forwardtotikztrue

```



```

6979 \forest@anchor@do{#1}{last'}{\forest@referencednodeid}%
6980 }
6981 \pgfdeclaregenericanchor{children last'}{%
6982 \forest@anchor@forwardtotikztrue
6983 \forest@anchor@do{#1}{children last'}{\forest@referencednodeid}%
6984 }

```

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

```

6985 \def\forest@anchor@do#1#2#3{% #1 = shape name, #2 = (potentially) forest anchor, #3 = node id
6986 \forest@for node{#3}{%
6987 \def\forest@temp@anchor{#2}%
6988 \forest@anchor@snapbordertocompassfalse
6989 \forest@anchor@isborderfalse
6990 \forest@anchor@to@tikz@anchor
6991 \forest@anchor@border@to@compass
6992 \ifforest@anchor@forwardtotikz
6993 \forest@anchor@forward{#1}%
6994 \else
6995 \fi
6996 }%
6997 }

```

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

```

6998 \def\forest@anchor@to@tikz@anchor{%
6999 \ifcsdef{forest@anchor@@\forest@temp@anchor}{%
7000 \csuse{forest@anchor@@\forest@temp@anchor}%
7001 \forest@anchor@to@tikz@anchor
7002 }{}%
7003 }

```

Actual computation.

```

7004 \csdef{forest@anchor@@parent anchor}{%
7005 \foresttoget{parent anchor}\forest@temp@anchor}
7006 \csdef{forest@anchor@@child anchor}{%
7007 \foresttoget{child anchor}\forest@temp@anchor}
7008 \csdef{forest@anchor@@anchor}{%
7009 \foresttoget{anchor}\forest@temp@anchor}
7010 \csdef{forest@anchor@@children'}{%
7011 \forest@anchor@isbordertrue
7012 \edef\forest@temp@anchor{\number\numexpr\forestove{grow}-\forestove{rotate}}}%
7013 }
7014 \csdef{forest@anchor@@parent'}{%
7015 \forest@anchor@isbordertrue
7016 \edef\forest@temp@anchor{\number\numexpr\forestove{grow}-\forestove{rotate}+180}%
7017 }
7018 \csdef{forest@anchor@@first'}{%
7019 \forest@anchor@isbordertrue
7020 \edef\forest@temp@anchor{\number\numexpr\forestove{grow}-\forestove{rotate}\ifnum\forestove{reversed}=0 -\e
7021 }
7022 \csdef{forest@anchor@@last'}{%
7023 \forest@anchor@isbordertrue
7024 \edef\forest@temp@anchor{\number\numexpr\forestove{grow}-\forestove{rotate}\ifnum\forestove{reversed}=0 +\e
7025 }
7026 \csdef{forest@anchor@@parent first'}{%
7027 \forest@anchor@isbordertrue
7028 \edef\forest@temp@anchor@parent{\number\numexpr\forestove{grow}-\forestove{rotate}+180}%
7029 \edef\forest@temp@anchor@first{\number\numexpr\forestove{grow}-\forestove{rotate}\ifnum\forestove{reversed}
7030 \forest@getaverageangle{\forest@temp@anchor@parent}{\forest@temp@anchor@first}\forest@temp@anchor
7031 }
7032 \csdef{forest@anchor@@parent last'}{%
7033 \forest@anchor@isbordertrue
7034 \edef\forest@temp@anchor@parent{\number\numexpr\forestove{grow}-\forestove{rotate}+180}%

```

```

7035 \edef\forest@temp@anchor@last{\number\numexpr\forestove{grow}-\forestove{rotate}\ifnum\forestove{reversed}=
7036 \forest@getaverageangle{\forest@temp@anchor@parent}{\forest@temp@anchor@last}\forest@temp@anchor
7037 }
7038 \csdef{forest@anchor@@children first'}{%
7039 \forest@anchor@isbordertrue
7040 \edef\forest@temp@anchor@first{\number\numexpr\forestove{grow}-\forestove{rotate}\ifnum\forestove{reversed}=
7041 \forest@getaverageangle{\forestove{grow}-\forestove{rotate}}{\forest@temp@anchor@first}\forest@temp@anchor
7042 }
7043 \csdef{forest@anchor@@children last'}{%
7044 \forest@anchor@isbordertrue
7045 \edef\forest@temp@anchor@last{\number\numexpr\forestove{grow}-\forestove{rotate}\ifnum\forestove{reversed}=
7046 \forest@getaverageangle{\forestove{grow}-\forestove{rotate}}{\forest@temp@anchor@last}\forest@temp@anchor
7047 }
7048 \csdef{forest@anchor@@children}{%
7049 \forest@anchor@snapbordertocompasstrue
7050 \csuse{forest@anchor@@children'}%
7051 }
7052 \csdef{forest@anchor@@parent}{%
7053 \forest@anchor@snapbordertocompasstrue
7054 \csuse{forest@anchor@@parent'}%
7055 }
7056 \csdef{forest@anchor@@first}{%
7057 \forest@anchor@snapbordertocompasstrue
7058 \csuse{forest@anchor@@first'}%
7059 }
7060 \csdef{forest@anchor@@last}{%
7061 \forest@anchor@snapbordertocompasstrue
7062 \csuse{forest@anchor@@last'}%
7063 }
7064 \csdef{forest@anchor@@parent first}{%
7065 \forest@anchor@snapbordertocompasstrue
7066 \csuse{forest@anchor@@parent first'}%
7067 }
7068 \csdef{forest@anchor@@parent last}{%
7069 \forest@anchor@snapbordertocompasstrue
7070 \csuse{forest@anchor@@parent last'}%
7071 }
7072 \csdef{forest@anchor@@children first}{%
7073 \forest@anchor@snapbordertocompasstrue
7074 \csuse{forest@anchor@@children first'}%
7075 }
7076 \csdef{forest@anchor@@children last}{%
7077 \forest@anchor@snapbordertocompasstrue
7078 \csuse{forest@anchor@@children last'}%
7079 }

```

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>.

```

7080 \def\forest@getaverageangle#1#2#3{%
7081 \edef\forest@temp{\number\numexpr #1-#2+540}%
7082 \pgfmathMod{\forest@temp}{360}\pgfmathtruncatemacro\pgfmathresult{\pgfmathresult}
7083 \edef\forest@temp{360+#2+((\pgfmathresult-180)/2)}%
7084 \pgfmathMod{\forest@temp}{360}\pgfmathtruncatemacro#3{\pgfmathresult}%
7085 }

```

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

```

7086 \def\forest@anchor@border@to@compass{%
7087 \ifforest@anchor@isborder
7088 \ifforest@anchor@snapbordertocompass

```

```

7089     \forest@anchor@snap@border@to@compass
7090 \else
7091     \pgfmathMod{\forest@temp@anchor}{360}%
7092     \pgfmathtruncatemacro\forest@temp@anchor{\pgfmathresult}%
7093 \fi
7094 \ifforest@anchor@forwardtotikz
7095     \ifcsname pgf@sh@ns@\pgfreferencednodename\endcsname
7096         @\csname pgf@sh@ns@\pgfreferencednodename\endcsname
7097         @\csname forest@compass@\forest@temp@anchor\endcsname
7098         \endcsname
7099         \letcs\forest@temp@anchor{forest@compass@\forest@temp@anchor}%
7100 \fi
7101 \else
7102     \ifforest@anchor@snapbordertocompass
7103         \letcs\forest@temp@anchor{forest@compass@\forest@temp@anchor}%
7104 \fi
7105 \fi
7106 \fi
7107 }
7108 \csdef{forest@compass@0}{east}
7109 \csdef{forest@compass@45}{north east}
7110 \csdef{forest@compass@90}{north}
7111 \csdef{forest@compass@135}{north west}
7112 \csdef{forest@compass@180}{west}
7113 \csdef{forest@compass@225}{south west}
7114 \csdef{forest@compass@270}{south}
7115 \csdef{forest@compass@315}{south east}
7116 \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).

```

7117 \def\forest@anchor@snap@border@to@compass{%
7118     \pgfmathMod{\forest@temp@anchor}{360}%
7119     \pgfmathdivide{\pgfmathresult}{45}%
7120     \pgfmathround{\pgfmathresult}%
7121     \pgfmathmultiply{\pgfmathresult}{45}%
7122     \pgfmathtruncatemacro\forest@temp@anchor{\pgfmathresult}%
7123 }

```

This macro forwards the resulting anchor to TikZ.

```

7124 \def\forest@anchor@forward#1{% #1 = shape name
7125     \ifdefempty\forest@temp@anchor{%
7126         \pgf@sh@reanchor{#1}{center}%
7127         \xdef\forest@hack@tikzshapeborder{%
7128             \noexpand\tikz@shapebordertrue
7129             \def\noexpand\tikz@shapeborder@name{\pgfreferencednodename}%
7130         }\aftergroup\forest@hack@tikzshapeborder
7131     }{%
7132         \pgf@sh@reanchor{#1}{\forest@temp@anchor}%
7133     }%
7134 }

```

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

```

7135 \def\forest@referencednodeid{\forest@node@Nametoid{\forest@referencednodename}}%
7136 \def\forest@referencednodename{%
7137     \expandafter\expandafter\expandafter\forest@referencednodename@\expandafter\pgfreferencednodename\forest@pgf
7138 }
7139 \expandafter\def\expandafter\forest@referencednodename@\expandafter#\expandafter1\forest@pgf@notyetpositioned
7140     \if\relax#1\relax\forest@referencednodename@stripafter#2\relax\fi
7141     \if\relax#2\relax#1\fi
7142 }
7143 \expandafter\def\expandafter\forest@referencednodename@stripafter\expandafter#\expandafter1\forest@pgf@notyet

```

This macro sets up `\pgf@x` and `\pgf@y` to the given anchor's coordinates, within the node's coordinate system. It works even before the node was positioned. If the anchor is empty, i.e. if is the implicit border anchor, we return the coordinates for the center.

```

7144 \def\forest@Pointanchor#1#2{% #1 = node id, #2 = anchor
7145   {%
7146     \def\forest@pa@temp@name{name}%
7147     \forest0ifdefined{#1}{@box}{%
7148       \forest0get{#1}{@box}\forest@temp
7149       \ifdefempty\forest@temp{%
7150         \def\forest@pa@temp@name{later@name}%
7151       }%
7152     }{%
7153       \setbox0\hbox{%
7154         \begin{pgfpicture}%
7155           \if\relax\forest0ve{#1}{#2}\relax
7156             \pgfpointanchor{\forest0ve{#1}{\forest@pa@temp@name}}{center}%
7157           \else
7158             \pgfpointanchor{\forest0ve{#1}{\forest@pa@temp@name}}{\forest0ve{#1}{#2}}%
7159           \fi
7160           \xdef\forest@global@marshal{%
7161             \noexpand\global\noexpand\pgf@x=\the\pgf@x\relax
7162             \noexpand\global\noexpand\pgf@y=\the\pgf@y
7163           }%
7164           \end{pgfpicture}%
7165         }%
7166       }%
7167       \forest@global@marshal
7168     }
7169 \def\forest@pointanchor#1{% #1 = anchor
7170   \forest@Pointanchor{\forest@cn}{#1}%
7171 }
```

10 Compatibility with previous versions

```

7172 \ifdefempty{\forest@compat}{%
7173   \RequirePackage{forest-compat}
7174 }
```