

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

v2.0.3

Sašo Živanović*

April 9, 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	31
6.3.1	Option-declaration mechanism	31
6.3.2	Registers	40
6.3.3	Declaring options	45
6.3.4	Option propagation	52
6.4	Aggregate functions	54
6.4.1	<code>pgfmath</code> extensions	55
6.5	Nodewalk	57
6.6	Dynamic tree	76
7	Stages	81
7.1	Typesetting nodes	84
7.2	Packing	86
7.2.1	Tiers	96
7.2.2	Node boundary	100
7.3	Compute absolute positions	105
7.4	Drawing the tree	105

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

8	Geometry	108
8.1	Projections	108
8.2	Break path	111
8.3	Get tight edge of path	113
8.4	Get rectangle/band edge	119
8.5	Distance between paths	120
8.6	Utilities	122
9	The outer UI	124
9.1	Externalization	124
9.2	The <code>forest</code> environment	125
9.3	Standard node	129
9.4	1s coordinate system	130
9.5	Relative node names in <code>TikZ</code>	131
9.6	Anchors	132
10	Compatibility with previous versions	137

1 Identification

```

1 \ProvidesPackage{forest}[2016/04/09 v2.0.3 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{\pgfkeys{/forest}{#1}}
```

2 Package options

```

19 \newif\ifforest@external@
20 \newif\ifforesttikzcshack
21 \newif\ifforest@install@keys@to@tikz@path@
22 \newif\ifforestdebugnodewalks
23 \newif\ifforestdebugdynamics
24 \def\forest@compat{}
25 \forestset{package@options/.cd,
26   external/.is if=forest@external@,
27   tikzcshack/.is if=foresttikzcshack,
28   tikzinstallkeys/.is if=forest@install@keys@to@tikz@path@,
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 debug/.code={\pgfqkeys{/forest/package@options/debug}{#1}},
38 debug/.default={nodewalks,dynamics},
39 debug/nodewalks/.is if=forestdebugnodewalks,
40 debug/dynamics/.is if=forestdebugdynamics,
41 }
42 \forest@install@keys@to@tikz@path@true
43 \foresttikz@hack@true
44 \def\forest@loadlibrarieslater{
45 \AtEndOfPackage{\forest@loadlibrarieslater}
46 \NewDocumentCommand\useforestlibrary{s O{}} m{%
47 \def\useforestlibrary@##1{\useforestlibrary@{#2}{##1}}%
48 \forcsvlist\useforestlibrary@{#3}%
49 \IfBooleanT{#1}{\forestapplylibrarydefaults{#3}}%
50 }
51 \def\useforestlibrary@#1#2{\RequirePackage[#1]{forest-lib-#2}}
52 \def\forestapplylibrarydefaults#1{\forcsvlist\forestapplylibrarydefaults@{#1}}
53 \def\forestapplylibrarydefaults@#1{\forestset{libraries/#1/defaults/.try}}
54 \NewDocumentCommand\ProvidesForestLibrary{m O{}}{\ProvidesPackage{forest-lib-#1}[#2]}
55 \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.)

```

56 \def\forest@patch#1#2#3#4#5{%
57 % #1 = cs to be patched
58 % #2 = purpose of the patch
59 % #3 = macro arguments
60 % #4 = original code
61 % #5 = patched code
62 \csdef{forest@original@#1}#3{#4}%
63 \csdef{forest@patched@#1}#3{#5}%
64 \ifcsequal{#1}{forest@original@#1}{%
65 \csletcs{#1}{forest@patched@#1}%
66 }{%
67 \ifcsequal{#1}{forest@patched@#1}{% all is good, the patch is in!
68 }{%
69 \PackageWarning{forest}{Failed patching '\expandafter\string\csname #1\endcsname'. Purpose of the patch
70 }%
71 }%
72 }

```

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

```

73 \forest@patch{pgfgettransform}{fix a leaking space}{#1}{%
74 \edef#1{{\pgf@pt@aa}{\pgf@pt@ab}{\pgf@pt@ba}{\pgf@pt@bb}{\the\pgf@pt@x}{\the\pgf@pt@y}}
75 }{%
76 \edef#1{{\pgf@pt@aa}{\pgf@pt@ab}{\pgf@pt@ba}{\pgf@pt@bb}{\the\pgf@pt@x}{\the\pgf@pt@y}}}%
77 }

```

4 Utilities

Escaping \ifs.

```

78 \long\def\@escapeif#1#2\fi{\fi#1}
79 \long\def\@escapeifif#1#2\fi#3\fi{\fi\fi#1}
80 \long\def\@escapeififif#1#2\fi#3\fi#4\fi{\fi\fi\fi#1}

```

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

```

81 \def\newloop#1{%

```

```

82 \count@=\escapechar
83 \escapechar=-1
84 \expandafter\newloop@parse@loopname\string#1\newloop@end
85 \escapechar=\count@
86 }%
87 {\lccode'7='1 \lccode'8='o \lccode'9='p
88 \lowercase{\gdef\newloop@parse@loopname#17889#2\newloop@end{%
89 \edef\newloop@marshal{%
90 \noexpand\csdef{#1loop#2}###1\expandafter\noexpand\csname #1repeat#2\endcsname{%
91 \noexpand\csdef{#1iterate#2}{###1\relax\noexpand\expandafter\expandafter\noexpand\csname#1iterate#
92 \expandafter\noexpand\csname#1iterate#2\endcsname
93 \let\expandafter\noexpand\csname#1iterate#2\endcsname\relax
94 }%
95 }%
96 \newloop@marshal
97 }%
98 }%
99 }%

```

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.

```

100 \def\newsafeloop#1{%
101 \csdef{safeloop@#1}##1\saferepeat{%
102 \edef\safeloop@marshal{%
103 \noexpand\csdef{safeiterate@#1}{%
104 \unexpanded{##1}\relax
105 \noexpand\expandafter
106 \expandonce{\csname safeiterate@#1\endcsname}%
107 \noexpand\fi
108 }%
109 }\safeloop@marshal
110 \csuse{safeiterate@#1}%
111 \advance\noexpand\safeloop@depth-1\relax
112 \cslet{safeiterate@#1}\relax
113 }%
114 }%
115 \newcount\safeloop@depth
116 \def\safeloop{%
117 \advance\safeloop@depth1
118 \ifcsdef{safeloop@the\safeloop@depth}{\expandafter\newsafeloop\expandafter{\the\safeloop@depth}}%
119 \csdef{safeloopn@the\safeloop@depth}{0}%
120 \csuse{safeloopn@the\safeloop@depth}%
121 \csedef{safeloopn@the\safeloop@depth}{\number\numexpr\csuse{safeloopn@the\safeloop@depth}+1}%
122 }
123 \let\saferepeat\fi
124 \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.

```

125 \def\newsafeRKloop#1{%
126 \csdef{safeRKloop@#1}##1\safeRKrepeat{%
127 \edef\safeRKloop@marshal{%
128 \noexpand\csdef{safeRKiterate@#1}{%
129 \unexpanded{##1}\relax
130 \noexpand\expandafter
131 \expandonce{\csname safeRKiterate@#1\endcsname}%
132 \noexpand\fi
133 }%
134 }\safeRKloop@marshal
135 \csuse{safeRKiterate@#1}%

```

```

136 \advance\noexpand\safeRKloop@depth-1\relax
137 \cslet{safeRKiterate@#1}\relax
138 }%
139 \expandafter\newif\csname ifsafeRKbreak@\the\safeRKloop@depth\endcsname
140 }%
141 \newcount\safeRKloop@depth
142 \def\safeRKloop{%
143 \advance\safeRKloop@depth1
144 \ifcsdef{safeRKloop@\the\safeRKloop@depth}{\expandafter\newsafeRKloop\expandafter{\the\safeRKloop@depth}}
145 \csdef{safeRKloopn@\the\safeRKloop@depth}{0}%
146 \csuse{safeRKbreak@\the\safeRKloop@depth false}%
147 \csuse{safeRKloop@\the\safeRKloop@depth}%
148 \csedef{safeRKloopn@\the\safeRKloop@depth}{\number\numexpr\csuse{safeRKloopn@\the\safeRKloop@depth}+1}%
149 }
150 \let\safeRKrepeat\fi
151 \def\safeRKloopn{\csuse{safeRKloopn@\the\safeRKloop@depth}}%

```

Additional loops (for embedding).

```

152 \newloop\forest@loop

```

New counters, dimens, ifs.

```

153 \newdimen\forest@temp@dimen
154 \newcount\forest@temp@count
155 \newcount\forest@n
156 \newif\ifforest@temp
157 \newcount\forest@temp@global@count
158 \newtoks\forest@temp@toks

```

Appending and prepending to token lists.

```

159 \def\etotoks#1#2{\edef\pot@temp{#2}\expandafter#1\expandafter{\pot@temp}}
160 \def\apptotoks#1#2{\expandafter#1\expandafter{\the#1#2}}
161 \long\def\lapptotoks#1#2{\expandafter#1\expandafter{\the#1#2}}
162 \def\eaaptotoks#1#2{\edef\pot@temp{#2}\expandafter\expandafter\expandafter#1\expandafter\expandafter\expandafter
163 \def\pretotoks#1#2{\toks@=#2}\expandafter\expandafter\expandafter#1\expandafter\expandafter\expandafter{\exp
164 \def\epretotoks#1#2{\edef\pot@temp{#2}\expandafter\expandafter\expandafter#1\expandafter\expandafter\expandafter
165 \def\gapptotoks#1#2{\expandafter\global\expandafter#1\expandafter{\the#1#2}}
166 \def\xapptotoks#1#2{\edef\pot@temp{#2}\expandafter\expandafter\expandafter\global\expandafter\expandafter\exp
167 \def\gpretotoks#1#2{\toks@=#2}\expandafter\expandafter\expandafter\global\expandafter\expandafter\expandafter
168 \def\xpretotoks#1#2{\edef\pot@temp{#2}\expandafter\expandafter\expandafter\global\expandafter\expandafter\exp

```

Expanding number arguments.

```

169 \def\expandnumberarg#1#2{\expandafter#1\expandafter{\number#2}}
170 \def\expandtwonumberargs#1#2#3{%
171 \expandafter\expandtwonumberargs@\expandafter#1\expandafter{\number#3}{#2}}
172 \def\expandtwonumberargs@#1#2#3{%
173 \expandafter#1\expandafter{\number#3}{#2}}
174 \def\expandthreenumberargs#1#2#3#4{%
175 \expandafter\expandthreenumberargs@\expandafter#1\expandafter{\number#4}{#2}{#3}}
176 \def\expandthreenumberargs@#1#2#3#4{%
177 \expandafter\expandthreenumberargs@@\expandafter#1\expandafter{\number#4}{#2}{#3}}
178 \def\expandthreenumberargs@@#1#2#3#4{%
179 \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.

```

180 \def\forest@convert@others@to@underscores#1#2{%
181 \def\forest@cotu@result{}}%
182 \forest@cotu@first#1\forest@end
183 \let#2\forest@cotu@result
184 }
185 \def\forest@cotu{%
186 \let\forest@cotu@have@num\forest@cotu@have@alpha
187 \futurelet\forest@cotu@nextchar\forest@cotu@checkforspace

```

```

188 }
189 \def\forest@cotu@first{%
190   \let\forest@cotu@have@num\forest@cotu@have@other
191   \futurelet\forest@cotu@nextchar\forest@cotu@checkforspace
192 }
193 \def\forest@cotu@checkforspace{%
194   \expandafter\ifx\space\forest@cotu@nextchar
195   \let\forest@cotu@next\forest@cotu@havespace
196   \else
197     \let\forest@cotu@next\forest@cotu@nospace
198   \fi
199   \forest@cotu@next
200 }
201 \def\forest@cotu@havespace#1{%
202   \appto\forest@cotu@result{_}%
203   \forest@cotu#1%
204 }
205 \def\forest@cotu@nospace{%
206   \ifx\forest@cotu@nextchar\forest@end
207     \@escapeif\@gobble
208   \else
209     \@escapeif\forest@cotu@nospaceB
210   \fi
211 }
212 \def\forest@cotu@nospaceB#1{%
213   \ifcat#1a%
214     \let\forest@cotu@next\forest@cotu@have@alpha
215   \else
216     \if!\ifnum9<1#1!\fi
217     \let\forest@cotu@next\forest@cotu@have@num
218   \else
219     \let\forest@cotu@next\forest@cotu@have@other
220   \fi
221   \fi
222   \forest@cotu@next#1%
223 }
224 \def\forest@cotu@have@alpha#1{%
225   \appto\forest@cotu@result{#1}%
226   \forest@cotu
227 }
228 \def\forest@cotu@have@other#1{%
229   \appto\forest@cotu@result{_}%
230   \forest@cotu
231 }

  Additional list macros.

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

```

```

247 }%
248 \def\forest@listedel@B|##1\forest@END{%|
249   \def#1{##1}%
250 }%
251 \expandafter\forest@listedel@A\expandafter|#1\forest@END%|
252 }
253 \forest@restorelistsepcatcode
    Strip (the first level of) braces from all the tokens in the argument.
254 \def\forest@strip@braces#1{%
255   \forest@strip@braces@A#1\forest@strip@braces@preend\forest@strip@braces@end
256 }
257 \def\forest@strip@braces@A#1#2\forest@strip@braces@end{%
258   #1\ifx\forest@strip@braces@preend#2\else\escapeif{\forest@strip@braces@A#2\forest@strip@braces@end}\fi
259 }

    Utilities dealing with pgfkeys.
260 \def\forest@copycommandkey#1#2{% copies command of #1 into #2
261   \pgfkeysifdefined{#1/.@cmd}{%
262     \PackageError{forest}{Key #1 is not a command key}{%
263     }%
264     \pgfkeysgetvalue{#1/.@cmd}\forest@temp
265     \pgfkeyslet{#2/.@cmd}\forest@temp
266     \pgfkeysifdefined{#1/.@args}{%
267       \pgfkeysgetvalue{#1/.@args}\forest@temp
268       \pgfkeyslet{#2/.@args}\forest@temp
269     }{}%
270     \pgfkeysifdefined{#1/.@body}{%
271       \pgfkeysgetvalue{#1/.@body}\forest@temp
272       \pgfkeyslet{#2/.@body}\forest@temp
273     }{}%
274     \pgfkeysifdefined{#1/.@body}{%
275       \pgfkeysgetvalue{#1/.@body}\forest@temp
276       \pgfkeyslet{#2/.@body}\forest@temp
277     }{}%
278     \pgfkeysifdefined{#1/.@def}{%
279       \pgfkeysgetvalue{#1/.@def}\forest@temp
280       \pgfkeyslet{#2/.@def}\forest@temp
281     }{}%
282 }
283 \forestset{
284   copy command key/.code 2 args={\forest@copycommandkey{#1}{#2}},
285   autoforward/.code 2 args={\forest@autoforward{#1}{#2={#1={##1}}}{true}},
286   autoforward'/.code 2 args={\forest@autoforward{#1}{#2=#1,#2={#1={##1}}}{true}},
287   Autoforward/.code 2 args={\forest@autoforward{#1}{#2}{true}},
288   autoforward register/.code 2 args={\forest@autoforward{#1}{#2={#1={##1}}}{false}},
289   autoforward register'/.code 2 args={\forest@autoforward{#1}{#2=#1,#2={#1={##1}}}{false}},
290   Autoforward register/.code 2 args={\forest@autoforward{#1}{#2}{false}},
291   copy command key@if it exists/.code 2 args={%
292     \pgfkeysifdefined{#1/.@cmd}{%
293       \forest@copycommandkey{#1}{#2}%
294     }{}%
295   },
296   unautoforward/.style={
297     typeout={unautoforwarding #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     copy command key@if it exists={/forest/autoforwarded #1+'}{/forest/#1+'},

```

```

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

```



```

366 % #1 = keys to remove (a keylist: an empty value means remove a key with any value)
367 % #2 = keylist
368 % #3 = toks cs for result
369 \forest@temp@toks{}%
370 \def\forestnovalue{\forestnovalue}%
371 \pgfqkeys{/forest/remove@key@installer}{#1}%
372 \let\forestnovalue\pgfkeysnovaluetext
373 \pgfqkeys{/forest/remove@key}{#2}%
374 \pgfqkeys{/forest/remove@key@uninstaller}{#1}%
375 #3\forest@temp@toks
376 }
377 \def\forest@remove@key@novalue{\forest@remove@key@novalue}%
378 \forestset{
379   remove@key@installer/.unknown/.code={% #1 = (outer) value
380     \def\forest@temp{#1}%
381     \ifx\forest@temp\pgfkeysnovalue@text
382       \pgfkeysdef{/forest/remove@key/\pgfkeyscurrentname}{}%
383     \else
384       \ifx\forest@temp\forestnovalue
385         \expandafter\forest@remove@key@installer@defwithvalue\expandafter{\pgfkeyscurrentname}{\pgfkeysnovalu
386       \else
387         \expandafter\forest@remove@key@installer@defwithvalue\expandafter{\pgfkeyscurrentname}{#1}%
388       \fi
389     \fi
390   },
391   remove@key/.unknown/.code={% #1 = (inner) value
392     \expandafter\apptotoks\expandafter\forest@temp@toks\expandafter{\pgfkeyscurrentname={#1},}%
393   },
394   remove@key@uninstaller/.unknown/.code={%
395     \pgfkeyslet{/forest/remove@key/\pgfkeyscurrentname/.@cmd}\@undefined},
396 }
397 \def\forest@remove@key@installer@defwithvalue#1#2{% #1=key name, #2 = outer value
398   \pgfkeysdef{/forest/remove@key/#1}{% ##1 = inner value
399     \def\forest@temp@outer{#2}%
400     \def\forest@temp@inner{##1}%
401     \ifx\forest@temp@outer\forest@temp@inner
402     \else
403       \apptotoks\forest@temp@toks{#1={##1},}%
404     \fi
405   }%
406 }

```

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

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

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`.

take only two arguments: the array bounds.

```

407 \def\forest@sort#1#2#3#4#5{%
408   \let\forest@sort@cmp#1\relax
409   \let\forest@sort@let#2\relax
410   \let\forest@sort@direction#3\relax
411   \forest@@sort{#4}{#5}%
412 }

```

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.

```

413 \def\forest@quicksort@minarraylength{10000}
414 \def\forest@@sort#1#2{%
415   \ifnum#1<#2\relax\@escapeif{%
416     \forest@sort@m=#2
417     \advance\forest@sort@m -#1
418     \ifnum\forest@sort@m>\forest@quicksort@minarraylength\relax\@escapeif{%
419       \forest@quicksort{#1}{#2}%
420     }\else\@escapeif{%
421       \forest@insertionsort{#1}{#2}%
422     }\fi
423   }\fi
424 }

```

Various counters and macros needed by the sorting algorithms.

```

425 \newcount\forest@sort@m\newcount\forest@sort@k\newcount\forest@sort@p
426 \def\forest@sort@ascending{>}
427 \def\forest@sort@descending{<}
428 \def\forest@sort@cmp{%
429   \PackageError{sort}{You must define forest@sort@cmp function before calling
430     sort}{The macro must take two arguments, indices of the array
431     elements to be compared, and return '=' if the elements are equal
432     and '>'/ '<' if the first is greater /less than the secong element.}%
433 }
434 \def\forest@sort@cmp@gt{\def\forest@sort@cmp@result{>}}
435 \def\forest@sort@cmp@lt{\def\forest@sort@cmp@result{<}}
436 \def\forest@sort@cmp@eq{\def\forest@sort@cmp@result{=}}
437 \def\forest@sort@let{%
438   \PackageError{sort}{You must define forest@sort@let function before calling
439     sort}{The macro must take two arguments, indices of the array:
440     element 2 must be copied onto element 1.}%
441 }

```

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

```

442 \newloop\forest@sort@loop
443 \newloop\forest@sort@loopA
444 \def\forest@quicksort#1#2{%

```

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

```

445   \forest@sort@m=#2
446   \advance\forest@sort@m -#1
447   \ifodd\forest@sort@m\relax\advance\forest@sort@m1 \fi
448   \divide\forest@sort@m 2
449   \advance\forest@sort@m #1

```

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

```

450   \forest@sort@cmp{#1}{#2}%
451   \if\forest@sort@cmp@result=%
452     \forest@sort@p=#1
453   \else
454     \if\forest@sort@cmp@result>%
455       \forest@sort@p=#1\relax
456     \else

```

```

457     \forest@sort@p=#2\relax
458   \fi
459   \forest@sort@cmp{\the\forest@sort@p}{\the\forest@sort@m}%
460   \if\forest@sort@cmp@result<%
461   \else
462     \forest@sort@p=\the\forest@sort@m
463   \fi
464 \fi

```

Exchange the pivot and the first element.

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

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

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

Loop through the list.

```

467 \forest@sort@k=#1\relax
468 \forest@sort@loop
469 \ifnum\forest@sort@k<#2\relax
470   \advance\forest@sort@k 1

```

Compare the pivot and the current element.

```
471 \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.

```

472 \ifx\forest@sort@direction\forest@sort@cmp@result
473   \advance\forest@sort@m 1
474   \forest@sort@xch{\the\forest@sort@m}{\the\forest@sort@k}
475 \fi
476 \forest@sort@repeat

```

Move the pivot element into its final position.

```
477 \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.

```

478 \forest@sort@k=\forest@sort@m
479 \advance\forest@sort@k -1
480 \advance\forest@sort@m 1
481 \edef\forest@sort@marshal{%
482   \noexpand\forest@sort@{#1}{\the\forest@sort@k}%
483   \noexpand\forest@sort@{\the\forest@sort@m}{#2}%
484 }%
485 \forest@sort@marshal
486 }
487 % We defines the item-exchange macro in terms of the (user-provided)
488 % array let macro.
489 % \begin{macrocode}
490 \def\forest@sort@aux{aux}
491 \def\forest@sort@xch#1#2{%
492   \forest@sort@let{\forest@sort@aux}{#1}%
493   \forest@sort@let{#1}{#2}%
494   \forest@sort@let{#2}{\forest@sort@aux}%
495 }

```

Insertion sort.

```

496 \def\forest@insertionsort#1#2{%
497   \forest@sort@m=#1
498   \edef\forest@insertionsort@low{#1}%
499   \forest@sort@loopA
500   \ifnum\forest@sort@m<#2
501     \advance\forest@sort@m 1

```

```

502 \forest@insertionsort@qbody
503 \forest@sort@repeatA
504 }
505 \newif\ifforest@insertionsort@loop
506 \def\forest@insertionsort@qbody{%
507 \forest@sort@let{\forest@sort@aux}{\the\forest@sort@m}%
508 \forest@sort@k\forest@sort@m
509 \advance\forest@sort@k -1
510 \forest@insertionsort@looptrue
511 \forest@sort@loop
512 \ifforest@insertionsort@loop
513 \forest@insertionsort@qbody
514 \forest@sort@repeat
515 \advance\forest@sort@k 1
516 \forest@sort@let{\the\forest@sort@k}{\forest@sort@aux}%
517 }
518 \def\forest@insertionsort@qbody{%
519 \forest@sort@cmp{\the\forest@sort@k}{\forest@sort@aux}%
520 \ifx\forest@sort@direction\forest@sort@cmp@result\relax
521 \forest@sort@p=\forest@sort@k
522 \advance\forest@sort@p 1
523 \forest@sort@let{\the\forest@sort@p}{\the\forest@sort@k}%
524 \advance\forest@sort@k -1
525 \ifnum\forest@sort@k<\forest@insertionsort@low\relax
526 \forest@insertionsort@loopfalse
527 \fi
528 \else
529 \forest@insertionsort@loopfalse
530 \fi
531 }

```

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

Compare numbers.

```

532 \def\forest@sort@cmpnumcs#1#2{%
533 \ifnum\csname#1\endcsname>\csname#2\endcsname\relax
534 \forest@sort@cmp@gt
535 \else
536 \ifnum\csname#1\endcsname<\csname#2\endcsname\relax
537 \forest@sort@cmp@lt
538 \else
539 \forest@sort@cmp@eq
540 \fi
541 \fi
542 }

```

Compare dimensions.

```

543 \def\forest@sort@cmpdimcs#1#2{%
544 \ifdim\csname#1\endcsname>\csname#2\endcsname\relax
545 \forest@sort@cmp@gt
546 \else
547 \ifdim\csname#1\endcsname<\csname#2\endcsname\relax
548 \forest@sort@cmp@lt
549 \else
550 \forest@sort@cmp@eq
551 \fi
552 \fi
553 }

```

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

```

554 \def\forest@sort@cmptwodimcs#1#2#3#4{%
555 \ifdim\csname#1\endcsname>\csname#3\endcsname\relax

```

```

556 \forest@sort@cmp@gt
557 \else
558 \ifdim\csname#1\endcsname<\csname#3\endcsname\relax
559 \forest@sort@cmp@lt
560 \else
561 \ifdim\csname#2\endcsname>\csname#4\endcsname\relax
562 \forest@sort@cmp@gt
563 \else
564 \ifdim\csname#2\endcsname<\csname#4\endcsname\relax
565 \forest@sort@cmp@lt
566 \else
567 \forest@sort@cmp@eq
568 \fi
569 \fi
570 \fi
571 \fi
572 }

```

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).

```

573 \def\forest@reversearray#1#2#3{%
574 \let\forest@sort@let#1%
575 \c@pgf@countc=#2
576 \c@pgf@countd=#3
577 \advance\c@pgf@countd -1
578 \safeloop
579 \ifnum\c@pgf@countc<\c@pgf@countd\relax
580 \forest@sort@xch{\the\c@pgf@countc}{\the\c@pgf@countd}%
581 \advance\c@pgf@countc 1
582 \advance\c@pgf@countd -1
583 \saferepeat
584 }

```

5 The bracket representation parser

5.1 The user interface macros

Settings.

```

585 \def\bracketset#1{\pgfqkeys{/bracket}{#1}}%
586 \bracketset{%
587 /bracket/.is family,
588 /handlers/.let/.style={\pgfkeyscurrentpath/.code={\let#1##1}},
589 opening bracket/.let=\bracket@openingBracket,
590 closing bracket/.let=\bracket@closingBracket,
591 action character/.let=\bracket@actionCharacter,
592 opening bracket=[,
593 closing bracket=],
594 action character,
595 new node/.code n args={3}{% #1=preamble, #2=node spec, #3=cs receiving the id
596 \forest@node@new#3%
597 \forest@set{#3}{given options}{\forest@contentto=\unexpanded{#2}}%
598 \ifblank{#1}{}%
599 \forest@set{preamble}{#1}%
600 }%
601 },
602 set afterthought/.code 2 args={% #1=node id, #2=afterthought
603 \ifblank{#2}{\forest@appto{#1}{given options}{,afterthought={#2}}}%
604 }
605 }

```

`\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.)

```
606 \newtoks\bracket@content
607 \newtoks\bracket@afterthought
608 \def\bracketParse#1#2={%
609   \def\bracketEndParsingHook{#1}%
610   \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.)

```
611 \bracket@content={}%
612 \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.

```
613 \let\bracket@state\bracket@state@starting
614 \bracket@ignorespacestrue
```

By default, don’t expand anything.

```
615 \bracket@expandtokensfalse
```

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

```
616 \def\bracket@parentNode{0}%
617 \def\bracket@rootNode{0}%
618 \def\bracket@newNode{0}%
619 \def\bracket@afterthoughtNode{0}%
```

Finally, we start the parser.

```
620 \bracket@Parse
621 }
```

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.

```
622 \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.

```
623 \def\bracket@Parse{%
624   \futurelet\bracket@next@token\bracket@Parse@checkForSpace
625 }
626 \def\bracket@Parse@checkForSpace{%
627   \expandafter\ifx\space\bracket@next@token\@escapeif{%
628     \ifbracket@ignorespaces\else
629       \bracket@haveSpacetrue
630     \fi
631     \expandafter\bracket@Parse\romannumeral-‘0%
632   }\else\@escapeif{%
633     \bracket@Parse@maybeexpand
634   }\fi
635 }
```

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`.

```
636 \newif\ifbracket@expandtokens
637 \def\bracket@Parse@maybeexpand{%
638   \ifbracket@expandtokens\@escapeif{%
639     \expandafter\bracket@Parse@peekAhead\romannumeral-‘0%
```

```

640 } \else \@escapeif{%
641   \bracket@Parse@peekAhead
642 } \fi
643 }

```

We then look ahead to see what's coming.

```

644 \def\bracket@Parse@peekAhead{%
645   \futurelet\bracket@next@token\bracket@Parse@checkForTeXGroup
646 }

```

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

```

647 \def\bracket@Parse@checkForTeXGroup{%
648   \ifx\bracket@next@token\bgroup%
649     \@escapeif{\bracket@Parse@appendGroup}%
650   \else
651     \@escapeif{\bracket@Parse@token}%
652   \fi
653 }

```

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.

```

654 \long\def\bracket@Parse@token#1{%
655   \ifx#1\bracket@openingBracket
656     \@escapeif{\bracket@Parse@openingBracketFound}%
657   \else
658     \@escapeif{%
659       \ifx#1\bracket@closingBracket
660         \@escapeif{\bracket@Parse@closingBracketFound}%
661       \else
662         \@escapeif{%
663           \ifx#1\bracket@actionCharacter
664             \@escapeif{\futurelet\bracket@next@token\bracket@Parse@actionCharacterFound}%
665           \else
666             \@escapeif{\bracket@Parse@appendToken#1}%
667           \fi
668         }%
669       \fi
670     }%
671   \fi
672 }

```

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

```

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

```

```

691 \or
692   \lapptotoks\bracket@afterthought{#1}%
693 \or
694   \lapptotoks\bracket@afterthought{#1}%
695 \fi
696 \bracket@ignorespacesfalse
697 \bracket@Parse
698 }
699 \def\bracket@Parse@appendGroup#1{%
700   \bracket@Parse@appendSpace
701   \ifcase\bracket@state\relax
702     \apptotoks\bracket@content{{#1}}%
703   \or
704     \apptotoks\bracket@afterthought{{#1}}%
705   \or
706     \apptotoks\bracket@afterthought{{#1}}%
707   \fi
708   \bracket@ignorespacesfalse
709   \bracket@Parse
710 }

```

Declare states.

```

711 \def\bracket@state@inContent{0}
712 \def\bracket@state@inAfterthought{1}
713 \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.)

```

714 \def\bracket@Parse@openingBracketFound{%
715   \bracket@haveSpacefalse
716   \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.

```

717   \@escapeif{%
718     \bracket@createNode
719     \ifnum\bracket@parentNode=0 \else
720       \forest@node@Append{\bracket@parentNode}{\bracket@newNode}%
721     \fi
722     \let\bracket@parentNode\bracket@newNode
723     \bracket@Parse
724   }%
725   \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.

```

726   \@escapeif{%
727     \bracket@addAfterthought
728     \let\bracket@state\bracket@state@inContent
729     \bracket@Parse
730   }%
731   \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.

```

732   \@escapeif{%
733     \let\bracket@state\bracket@state@inContent

```



```

734     \bracket@Parse
735   }%
736 \fi
737 }
738 \def\bracket@Parse@closingBracketFound{%
739   \bracket@haveSpacefalse
740   \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.

```

741   \@escapeif{%
742     \bracket@createNode
743     \ifnum\bracket@parentNode=0
744       \@escapeif\bracket@endParsingHook
745     \else
746       \@escapeif{%
747         \let\bracket@afterthoughtNode\bracket@newNode
748         \let\bracket@state\bracket@state@inAfterthought
749         \forest@node@Append{\bracket@parentNode}{\bracket@newNode}%
750         \bracket@Parse
751       }%
752     \fi
753   }%
754 \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.

```

755   \@escapeif{%
756     \bracket@addAfterthought
757     \let\bracket@afterthoughtNode\bracket@parentNode
758     \edef\bracket@parentNode{\forestOve{\bracket@parentNode}{@parent}}%
759     \ifnum\bracket@parentNode=0
760       \expandafter\bracket@endParsingHook
761     \else
762       \expandafter\bracket@Parse
763     \fi
764   }%
765 \else % starting

```

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

```

766   \PackageError{forest}{You're attempting to start a bracket representation
767     with a closing bracket}{}%
768 \fi
769 }

```

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

```

770 \def\bracket@Parse@actionCharacterFound{%
  If a braced expression follows, its contents will be fully expanded.
771 \ifx\bracket@next@token\bgroup\@escapeif{%
772   \bracket@Parse@action@expandgroup
773 } \else \@escapeif{%
774   \bracket@Parse@action@notagroup
775 } \fi
776 }
777 \def\bracket@Parse@action@expandgroup#1{%
778   \edef\bracket@Parse@action@expandgroup@macro{#1}%
779   \expandafter\bracket@Parse\bracket@Parse@action@expandgroup@macro

```

```

780 }
781 \let\bracket@action@fullyexpandCharacter+
782 \let\bracket@action@dontexpandCharacter-
783 \let\bracket@action@executeCharacter!
784 \def\bracket@Parse@action@notagroup#1{%

```

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

```

785 \ifx#1\bracket@action@fullyexpandCharacter\@escapeif{%
786 \bracket@expandtokenstrue\bracket@Parse
787 }\else\@escapeif{%

```

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

```

788 \ifx#1\bracket@action@dontexpandCharacter\@escapeif{%
789 \bracket@expandtokensfalse\bracket@Parse
790 }\else\@escapeif{%

```

Inhibit expansion of the next token.

```

791 \ifx#10\@escapeif{%
792 \bracket@Parse@appendToken
793 }\else\@escapeif{%

```

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

```

794 \ifx#1\bracket@actionCharacter
795 \else\@escapeif{%

```

Anything else will be expanded once.

```

796 \expandafter\bracket@Parse#1%
797 }\fi
798 }\fi
799 }\fi
800 }\fi
801 }

```

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.

```

802 \def\bracket@createNode{%
803 \ifnum\bracket@rootNode=0
804 % root node
805 \bracketset{new node/.expanded=%
806 {\the\bracket@afterthought}%
807 {\the\bracket@content}%
808 \noexpand\bracket@newNode
809 }%
810 \bracket@afterthought={}%
811 \let\bracket@rootNode\bracket@newNode
812 \expandafter\let\bracket@saveRootNodeTo\bracket@newNode
813 \else
814 % other nodes
815 \bracketset{new node/.expanded=%
816 {}%
817 {\the\bracket@content}%
818 \noexpand\bracket@newNode
819 }%
820 \fi
821 \bracket@content={}%
822 }

```

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

```

823 \def\bracket@addAfterthought{%
824   \bracketset{%
825     set afterthought/.expanded={\bracket@afterthoughtNode}{\the\bracket@afterthought}%
826   }%
827   \bracket@afterthought={}%
828 }

```

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.

```

829 % full expansion expands precisely to the value
830 \def\forestov#1{\expandafter\expandafter\expandafter\expandonce
831   \pgfkeysvalueof{/forest/@node/\forest@cn/#1}}
832 % full expansion expands all the way
833 \def\forestove#1{\pgfkeysvalueof{/forest/@node/\forest@cn/#1}}
834 % full expansion expands to the cs holding the value
835 \def\forestom#1{\expandafter\expandonce\expandafter{\pgfkeysvalueof{/forest/@node/\forest@cn/#1}}}
836 \def\forestoget#1#2{\pgfkeysgetvalue{/forest/@node/\forest@cn/#1}{#2}}
837 \def\forestolet#1#2{\pgfkeyslet{/forest/@node/\forest@cn/#1}{#2}}
838 \def\forestocslet#1#2{%
839   \edef\forest@marshal{%
840     \noexpand\pgfkeyslet{/forest/@node/\forest@cn/#1}{\expandonce{\csname#2\endcsname}}%
841   }\forest@marshal
842 }
843 \def\forestoset#1#2{\pgfkeyssetvalue{/forest/@node/\forest@cn/#1}{#2}}
844 \def\forestoeset#1#2{%
845   \edef\forest@option@temp{%
846     \noexpand\pgfkeyssetvalue{/forest/@node/\forest@cn/#1}{#2}%
847   }\forest@option@temp
848 }
849 \def\forestoaappto#1#2{%
850   \forestoeset{#1}{\forestov{#1}\unexpanded{#2}}%
851 }
852 \def\forestoiddefined#1#2#3{%
853   \pgfkeysifdefined{/forest/@node/\forest@cn/#1}{#2}{#3}%
854 }

```

User macros for retrieving node options of the current node.

```

855 \let\forestoption\forestov
856 \let\foresteooption\forestove

```

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

```

857 \def\forest0v#1#2{\expandafter\expandafter\expandafter\expandonce
858   \pgfkeysvalueof{/forest/@node/#1/#2}}
859 \def\forest0ve#1#2{\pgfkeysvalueof{/forest/@node/#1/#2}}
860 % full expansion expands to the cs holding the value
861 \def\forest0m#1#2{\expandafter\expandonce\expandafter{\pgfkeysvalueof{/forest/@node/#1/#2}}}
862 \def\forest0get#1#2#3{\pgfkeysgetvalue{/forest/@node/#1/#2}{#3}}
863 \def\forest0get#1#2#3{\pgfkeysgetvalue{/forest/@node/#1/#2}{#3}}
864 \def\forest0let#1#2#3{\pgfkeyslet{/forest/@node/#1/#2}{#3}}
865 \def\forest0cslet#1#2#3{%
866   \edef\forest@marshal{%
867     \noexpand\pgfkeyslet{/forest/@node/#1/#2}{\expandonce{\csname#3\endcsname}}%

```

```

868 }\forest@marshal
869 }
870 \def\forestOset#1#2#3{\pgfkeyssetvalue{/forest/@node/#1/#2}{#3}}
871 \def\forestOeset#1#2#3{%
872   \edef\forestoption@temp{%
873     \noexpand\pgfkeyssetvalue{/forest/@node/#1/#2}{#3}%
874   }\forestoption@temp
875 }
876 \def\forestOappto#1#2#3{%
877   \forestOeset{#1}{#2}{\forestOv{#1}{#2}\unexpanded{#3}}%
878 }
879 \def\forestOeappto#1#2#3{%
880   \forestOeset{#1}{#2}{\forestOv{#1}{#2}#3}%
881 }
882 \def\forestOpreto#1#2#3{%
883   \forestOeset{#1}{#2}{\unexpanded{#3}\forestOv{#1}{#2}}%
884 }
885 \def\forestOepreto#1#2#3{%
886   \forestOeset{#1}{#2}{#3\forestOv{#1}{#2}}%
887 }
888 \def\forestOifdefined#1#2#3#4{%
889   \pgfkeysifdefined{/forest/@node/#1/#2}{#3}{#4}%
890 }
891 \def\forestOlet0#1#2#3#4{% option #2 of node #1 <-- option #4 of node #3
892   \forestOget{#3}{#4}\forestoption@temp
893   \forestOlet{#1}{#2}\forestoption@temp}
894 \def\forestOleto#1#2#3{%
895   \forestOget{#3}\forestoption@temp
896   \forestOlet{#1}{#2}\forestoption@temp}
897 \def\forestOlet0#1#2#3{%
898   \forestOget{#2}{#3}\forestoption@temp
899   \forestOlet{#1}\forestoption@temp}
900 \def\forestOleto#1#2{%
901   \forestOget{#2}\forestoption@temp
902   \forestOlet{#1}\forestoption@temp}

```

Macros for retrieving/setting registers.

```

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

```

```

927 \forestreset{#1}{\unexpanded{#2}\forestrv{#1}}%
928 }
929 \def\forestrifdefined#1#2#3{%
930 \pgfkeysifdefined{/forest/@node/register/#1}{#2}{#3}%
931 }

User macros for retrieving node options of the current node.
932 \def\forestregister#1{\forestrv{#1}}
933 \def\foresteregister#1{\forestrve{#1}}

Node initialization. Node option declarations append to \forest@node@init.
934 \def\forest@node@init{%
935 \forestoset{@parent}{0}%
936 \forestoset{@previous}{0}% previous sibling
937 \forestoset{@next}{0}% next sibling
938 \forestoset{@first}{0}% primary child
939 \forestoset{@last}{0}% last child
940 }
941 \def\forest@node@init#1{%
942 \pgfkeysgetvalue{/forest/#1}\forest@node@init@temp
943 \forest@node@init@temp
944 }
945 \newcount\forest@node@maxid
946 \def\forest@node@new#1{% #1 = cs receiving the new node id
947 \advance\forest@node@maxid1
948 \forest@node@init
949 \forest@node@init
950 \forest@node@init
951 \forest@node@init
952 \forest@node@init
953 \edef#1{\forest@node@init}%
954 }%
955 }
956 \let\forest@node@init\forest@node@init
957 \def\forest@node@copy#1#2{% #1=from node id, #2=cs receiving the new node id
958 \advance\forest@node@maxid1
959 \def\forest@node@copy##1{\ifstrequal{##1}{name}{\forest@node@copy{name}{\forest@node@copy{#1}{#2}}}{\forest@node@copy{#1}{#2}}}%
960 \forest@node@copy{#1}{#2}%
961 \forest@node@copy{#1}{#2}%
962 \forest@node@copy{#1}{#2}%
963 \forest@node@copy{#1}{#2}%
964 \edef#2{\forest@node@copy{#1}{#2}}%
965 }%
966 \let\forest@node@copy\forest@node@copy
967 }
968 \forestset{
969 copy name template/.code={\def\forest@node@copy@name@template##1{#1}},
970 copy name template/.default={\forest@node@copy@name@template},
971 copy name template
972 }
973 \def\forest@tree@copy#1#2{% #1=from node id, #2=cs receiving the new node id
974 \forest@node@copy{#1}{#2}%
975 \forest@node@copy{#1}{#2}%
976 \forest@node@copy{#1}{#2}%
977 \edef#2{\forest@node@copy{#1}{#2}}%
978 }%
979 }
980 \def\forest@tree@copy@#1#2{%
981 \forest@node@Foreachchild{#2}{%
982 \forest@node@copy{#1}{#2}%
983 \forest@node@copy{#1}{#2}%
984 }%

```

985 }

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.

986 `\def\forest@cn{0}`

987 `\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.

```

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

```

1035 \def\forest@node@Append@#1#2{%
1036   \ifnum\forest@ve{#2}{@parent}=0
1037   \else
1038     \PackageError{forest}{Append(#1,#2):
1039       node #2 already has a parent (\forest@ve{#2}{@parent})}{}%
1040   \fi
1041   \forest@set{#2}{@parent}{#1}%
1042   \forest@get{#1}{@last}\forest@node@temp
1043   \forest@set{#1}{@last}{#2}%
1044   \forest@set{#2}{@previous}{\forest@node@temp}%
1045   \ifnum\forest@node@temp=0
1046     \forest@set{#1}{@first}{#2}%
1047   \else
1048     \forest@set{\forest@node@temp}{@next}{#2}%
1049   \fi
1050 }
1051 \def\forest@node@Remove@#1{%
1052   \forest@get{#1}{@parent}\forest@node@temp@parent
1053   \ifnum\forest@node@temp@parent=0
1054   \else
1055     \forest@get{#1}{@previous}\forest@node@temp@previous
1056     \forest@get{#1}{@next}\forest@node@temp@next
1057     \ifnum\forest@node@temp@previous=0
1058       \forest@set{\forest@node@temp@parent}{@first}{\forest@node@temp@next}%
1059     \else
1060       \forest@set{\forest@node@temp@previous}{@next}{\forest@node@temp@next}%
1061     \fi
1062     \ifnum\forest@node@temp@next=0
1063       \forest@set{\forest@node@temp@parent}{@last}{\forest@node@temp@previous}%
1064     \else
1065       \forest@set{\forest@node@temp@next}{@previous}{\forest@node@temp@previous}%
1066     \fi
1067     \forest@set{#1}{@parent}{0}%
1068     \forest@set{#1}{@previous}{0}%
1069     \forest@set{#1}{@next}{0}%
1070   \fi
1071 }

```

Do some stuff and return to the current node.

```

1072 \def\forest@forthis#1{%
1073   \edef\forest@node@marshal{\unexpanded{#1}\def\noexpand\forest@cn}%
1074   \expandafter\forest@node@marshal\expandafter{\forest@cn}%
1075 }
1076 \def\forest@fornode#1#2{%
1077   \edef\forest@node@marshal{\edef\noexpand\forest@cn{#1}\unexpanded{#2}\def\noexpand\forest@cn}%
1078   \expandafter\forest@node@marshal\expandafter{\forest@cn}%
1079 }

```

Looping methods: children.

```

1080 \def\forest@node@foreachchild#1{\forest@node@Foreachchild{\forest@cn}{#1}}
1081 \def\forest@node@Foreachchild#1#2{%
1082   \forest@fornode{\forest@ve{#1}{@first}}{\forest@node@@forselfandfollowingsiblings{#2}}%
1083 }
1084 \def\forest@node@@forselfandfollowingsiblings#1{%
1085   \ifnum\forest@cn=0
1086   \else
1087     \forest@forthis{#1}%
1088     \@escapeif{%
1089       \edef\forest@cn{\forest@ve{@next}}%
1090       \forest@node@@forselfandfollowingsiblings{#1}%
1091     }%
1092   \fi

```

```

1093 }
1094 \def\forest@node@@forselfandfollowingsiblings@reversed#1{%
1095   \ifnum\forest@cn=0
1096   \else
1097     \@escapeif{%
1098       \edef\forest@marshal{%
1099         \noexpand\def\noexpand\forest@cn{\forestove{@next}}%
1100         \noexpand\forest@node@@forselfandfollowingsiblings@reversed{\unexpanded{#1}}%
1101         \noexpand\forest@fornode{\forest@cn}{\unexpanded{#1}}%
1102       }\forest@marshal
1103     }%
1104   \fi
1105 }
1106 \def\forest@node@foreachchild@reversed#1{\forest@node@Foreachchild@reversed{\forest@cn}{#1}}
1107 \def\forest@node@Foreachchild@reversed#1#2{%
1108   \forest@fornode{\forestOve{#1}{@last}}{\forest@node@@forselfandprecedingsiblings@reversed{#2}}%
1109 }
1110 \def\forest@node@@forselfandprecedingsiblings@reversed#1{%
1111   \ifnum\forest@cn=0
1112   \else
1113     \forest@forthis{#1}%
1114     \@escapeif{%
1115       \edef\forest@cn{\forestove{@previous}}%
1116       \forest@node@@forselfandprecedingsiblings@reversed{#1}%
1117     }%
1118   \fi
1119 }
1120 \def\forest@node@@forselfandprecedingsiblings#1{%
1121   \ifnum\forest@cn=0
1122   \else
1123     \@escapeif{%
1124       \edef\forest@marshal{%
1125         \noexpand\def\noexpand\forest@cn{\forestove{@previous}}%
1126         \noexpand\forest@node@@forselfandprecedingsiblings{\unexpanded{#1}}%
1127         \noexpand\forest@fornode{\forest@cn}{\unexpanded{#1}}%
1128       }\forest@marshal
1129     }%
1130   \fi
1131 }

```

Looping methods: (sub)tree and descendants.

```

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

```



```

1152 \def\forest@node@foreach@reversed#1{%
1153   \forest@node@@foreach{#1}{-1}{@last}{\forest@node@@forselfandprecedingsiblings@reversed}}
1154 \def\forest@node@Foreach@reversed#1#2{%
1155   \forest@fornode{#1}{\forest@node@@foreach{#2}{-1}{@last}{\forest@node@@forselfandprecedingsiblings@reversed}}
1156 \def\forest@node@foreach@childrenfirst#1{%
1157   \forest@node@@foreach{#1}{1}{@first}{\forest@node@@forselfandfollowingsiblings}}
1158 \def\forest@node@Foreach@childrenfirst#1#2{%
1159   \forest@fornode{#1}{\forest@node@@foreach{#2}{1}{@first}{\forest@node@@forselfandfollowingsiblings}}}
1160 \def\forest@node@foreach@childrenfirst@reversed#1{%
1161   \forest@node@@foreach{#1}{1}{@last}{\forest@node@@forselfandprecedingsiblings@reversed}}
1162 \def\forest@node@Foreach@childrenfirst@reversed#1#2{%
1163   \forest@fornode{#1}{\forest@node@@foreach{#2}{1}{@last}{\forest@node@@forselfandprecedingsiblings@reversed}}}
1164 \def\forest@node@foreach@descendant#1{%
1165   \forest@node@foreachchild{\forest@node@@foreach{#1}{-1}{@first}{\forest@node@@forselfandfollowingsiblings}}
1166 \def\forest@node@Foreach@descendant#1#2{%
1167   \forest@node@Foreachchild{#1}{\forest@node@@foreach{#2}{-1}{@first}{\forest@node@@forselfandfollowingsiblings}}}
1168 \def\forest@node@foreach@descendant@reversed#1{%
1169   \forest@node@foreachchild@reversed{\forest@node@@foreach{#1}{-1}{@last}{\forest@node@@forselfandprecedingsiblings@reversed}}
1170 \def\forest@node@Foreach@descendant@reversed#1#2{%
1171   \forest@node@Foreachchild@reversed{#1}{\forest@node@@foreach{#2}{-1}{@last}{\forest@node@@forselfandprecedingsiblings@reversed}}}
1172 \def\forest@node@foreach@descendant@childrenfirst#1{%
1173   \forest@node@foreachchild{\forest@node@@foreach{#1}{1}{@first}{\forest@node@@forselfandfollowingsiblings}}}
1174 \def\forest@node@Foreach@descendant@childrenfirst#1#2{%
1175   \forest@node@Foreachchild{#1}{\forest@node@@foreach{#2}{1}{@first}{\forest@node@@forselfandfollowingsiblings}}}
1176 \def\forest@node@foreach@descendant@childrenfirst@reversed#1{%
1177   \forest@node@foreachchild@reversed{\forest@node@@foreach{#1}{1}{@last}{\forest@node@@forselfandprecedingsiblings@reversed}}
1178 \def\forest@node@Foreach@descendant@childrenfirst@reversed#1#2{%
1179   \forest@node@Foreachchild@reversed{#1}{\forest@node@@foreach{#2}{1}{@last}{\forest@node@@forselfandprecedingsiblings@reversed}}}

```

Looping methods: breadth-first.

```

1180 \def\forest@node@foreach@breadthfirst#1#2{% #1 = max level, #2 = code
1181   \forest@node@Foreach@breadthfirst@{\forest@cn}{@first}{@next}{#1}{#2}}
1182 \def\forest@node@foreach@breadthfirst@reversed#1#2{% #1 = max level, #2 = code
1183   \forest@node@Foreach@breadthfirst@{\forest@cn}{@last}{@previous}{#1}{#2}}
1184 \def\forest@node@Foreach@breadthfirst#1#2#3{% #1 = node id, #2 = max level, #3 = code
1185   \forest@node@Foreach@breadthfirst@{#1}{@first}{@next}{#2}{#3}}
1186 \def\forest@node@Foreach@breadthfirst@reversed#1#2#3{% #1 = node id, #2 = max level, #3 = code
1187   \forest@node@Foreach@breadthfirst@{#1}{@last}{@previous}{#2}{#3}}
1188 \def\forest@node@Foreach@breadthfirst@#1#2#3#4#5{%
1189   % #1 = root node,
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   \forest@node@Foreach@breadthfirst@processqueue{#1},{#2}{#3}{#4}{#5}%
1194 }
1195 \def\forest@node@Foreach@breadthfirst@processqueue#1#2#3#4#5{%
1196   % #1 = queue,
1197   % #2 = @first/@last, #3 = @next/@previous (must be in sync with #2),
1198   % #4 = max level (< 0 means infinite)
1199   % #5 = code to execute at each node
1200   \ifstrempy{#1}{}{%
1201     \forest@node@Foreach@breadthfirst@processqueue{#1}\forest@node@Foreach@breadthfirst@processqueue@
1202       {#2}{#3}{#4}{#5}%
1203   }%
1204 }
1205 \def\forest@node@Foreach@breadthfirst@processqueue#1,#2\forest@node@Foreach@breadthfirst@processqueue@#3#4#5{%
1206   % #1 = first,
1207   % #2 = rest,
1208   % #3 = @first/@last, #4 = next/previous (must be in sync with #2),
1209   % #5 = max level (< 0 means infinite)
1210   % #6 = code to execute at each node

```

```

1211 \forest@fornode{#1}{%
1212   #6%
1213   \ifnum#5<0
1214     \forest@node@getlistofchildren\forest@temp{#3}{#4}%
1215   \else
1216     \ifnum\forest@ve{level}>#5\relax
1217     \def\forest@temp{%
1218       \else
1219       \forest@node@getlistofchildren\forest@temp{#3}{#4}%
1220     \fi
1221   \fi
1222   \edef\forest@marshal{%
1223     \noexpand\forest@node@Foreach@breadthfirst@processqueue{\unexpanded{#2}\forest@temp}%
1224     {#3}{#4}{#5}{\unexpanded{#6}}%
1225   }\forest@marshal
1226 }%
1227 }
1228 \def\forest@node@getlistofchildren#1#2#3{% #1 = list cs, #2 = @first/@last, #3 = @next/@previous
1229 \forest@node@Getlistofchildren{\forest@cn}{#1}{#2}{#3}%
1230 }
1231 \def\forest@node@Getlistofchildren#1#2#3#4{% #1 = node, #2 = list cs, #3 = @first/@last, #4 = @next/@previous
1232 \def#2{%
1233 \ifnum\forest@ve{#3}=0
1234 \else
1235 \eappto#2{\forest@ve{#1}{#3}},}%
1236 \@escapeif{%
1237 \edef\forest@marshal{%
1238 \noexpand\forest@node@Getlistofchildren@{\forest@ve{#1}{#3}}\noexpand#2{#4}%
1239 }\forest@marshal
1240 }%
1241 \fi
1242 }
1243 \def\forest@node@Getlistofchildren@#1#2#3{% #1 = node, #2 = list cs, #3 = @next/@previous
1244 \ifnum\forest@ve{#1}{#3}=0
1245 \else
1246 \eappto#2{\forest@ve{#1}{#3}},}%
1247 \@escapeif{%
1248 \edef\forest@marshal{%
1249 \noexpand\forest@node@Getlistofchildren@{\forest@ve{#1}{#3}}\noexpand#2{#3}%
1250 }\forest@marshal
1251 }%
1252 \fi
1253 }

```

Compute n, n', n children and level.

```

1254 \def\forest@node@Compute@numeric@ts@info@#1{%
1255 \forest@node@Foreach{#1}{\forest@node@@compute@numeric@ts@info}%
1256 \ifnum\forest@ve{#1}{@parent}=0
1257 \else
1258 \forest@fornode{#1}{\forest@node@@compute@numeric@ts@info@nbar}%
1259 % hack: the parent of the node we called the update for gets +1 for n_children
1260 \edef\forest@node@temp{\forest@ve{#1}{@parent}}%
1261 \forest@set{\forest@node@temp}{n children}{%
1262 \number\numexpr\forest@ve{\forest@node@temp}{n children}-1%
1263 }%
1264 \fi
1265 \forest@node@Foreachdescendant{#1}{\forest@node@@compute@numeric@ts@info@nbar}%
1266 }
1267 \def\forest@node@@compute@numeric@ts@info{%
1268 \forest@set{n children}{0}%
1269 %

```

```

1270 \edef\forest@node@temp{\forestove{@previous}}%
1271 \ifnum\forest@node@temp=0
1272   \forestoset{n}{1}%
1273 \else
1274   \forestoeset{n}{\number\numexpr\forestOve{\forest@node@temp}{n}+1}%
1275 \fi
1276 %
1277 \edef\forest@node@temp{\forestove{@parent}}%
1278 \ifnum\forest@node@temp=0
1279   \forestoset{n}{0}%
1280   \forestoset{n'}{0}%
1281   \forestoset{level}{0}%
1282 \else
1283   \forestOeset{\forest@node@temp}{n children}{%
1284     \number\numexpr\forestOve{\forest@node@temp}{n children}+1%
1285   }%
1286   \forestoeset{level}{%
1287     \number\numexpr\forestOve{\forest@node@temp}{level}+1%
1288   }%
1289 \fi
1290 }
1291 \def\forest@node@@compute@numeric@ts@info@nbar{%
1292   \forestoeset{n'}{\number\numexpr\forestOve{\forestove{@parent}}{n children}-\forestove{n}+1}%
1293 }
1294 \def\forest@node@compute@numeric@ts@info#1{%
1295   \expandnumberarg\forest@node@Compute@numeric@ts@info@{\forest@cn}%
1296 }
1297 \def\forest@node@Compute@numeric@ts@info#1{%
1298   \expandnumberarg\forest@node@Compute@numeric@ts@info@{#1}%
1299 }

```

Tree structure queries.

```

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

```

```

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

```

```

1390 \fi
1391 }
1392 \def\forest@node@nextleafid@Goup#1{%
1393 \ifnum\forestOve{#1}{@parent}=0
1394 \PackageError{forest}{get next leaf: this is the last leaf}{}%
1395 \else
1396 \@escapeif{\expandnumberarg\forest@node@Nextleafid{\forestOve{#1}{@parent}}}%
1397 \fi
1398 }
1399 \def\forest@node@nextleafid@Godown#1{%
1400 \ifnum\forestOve{#1}{@first}=0
1401 #1%
1402 \else
1403 \@escapeif{\expandnumberarg\forest@node@nextleafid@Godown{\forestOve{#1}{@first}}}%
1404 \fi
1405 }
1406
1407
1408
1409 \def\forest@node@linearnextid{%
1410 \ifnum\forestove{@first}=0
1411 \expandafter\forest@node@linearnextnotdescendantid
1412 \else
1413 \forestove{@first}%
1414 \fi
1415 }
1416 \def\forest@node@linearnextnotdescendantid{%
1417 \expandnumberarg\forest@node@Linearnextnotdescendantid{\forest@cn}%
1418 }
1419 \def\forest@node@Linearnextnotdescendantid#1{%
1420 \ifnum\forestOve{#1}{@next}=0
1421 \ifnum\forestOve{#1}{@parent}=0
1422 0%
1423 \else
1424 \@escapeifif{\expandnumberarg\forest@node@Linearnextnotdescendantid{\forestOve{#1}{@parent}}}%
1425 \fi
1426 \else
1427 \forestOve{#1}{@next}%
1428 \fi
1429 }
1430 \def\forest@node@linearpreviousid{%
1431 \ifnum\forestove{@previous}=0
1432 \forestove{@parent}%
1433 \else
1434 \forest@node@previousleafid
1435 \fi
1436 }

```

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

```

1437
1438 \def\forest@ifancestorof#1{% is the current node an ancestor of #1? Yes: #2, no: #3
1439 \begingroup
1440 \expandnumberarg\forest@ifancestorof@{\forestOve{#1}{@parent}}%
1441 }
1442 \def\forest@ifancestorof@#1{%
1443 \ifnum#1=0
1444 \def\forest@ifancestorof@next{\expandafter\endgroup\@secondoftwo}%
1445 \else
1446 \ifnum\forest@cn=#1

```

```

1447 \def\forest@ifancestorof@next{\expandafter\endgroup\@firstoftwo}%
1448 \else
1449 \ifcsdef{forest@circularity@used#1}{%
We have just detected circularity: the potential descendant is in fact an ancestor of itself. Our answer
is “false”: the current node is not an ancestor of the potential descendant.
1450 \def\forest@ifancestorof@next{\expandafter\endgroup\@secondoftwo}%
1451 }{%
1452 \csdef{forest@circularity@used#1}{}%
1453 \def\forest@ifancestorof@next{\expandnumberarg\forest@ifancestorof@{\forestOve{#1}{@parent}}}%
1454 }%
1455 \fi
1456 \fi
1457 \forest@ifancestorof@next
1458 }

```

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

```

1459 \NewDocumentCommand\forestdebugtypeouttrees{o}{%
1460 \forestdebug@typeouttrees\forest@temp
1461 \typeout{\IfValueTF{#1}{#1: }{\forest@temp}}%
1462 }
1463 \def\forestdebug@typeouttrees#1{% #1 = cs to store the result
1464 \begingroup
1465 \edef\forest@temp@message{%
1466 \def\forestdebug@typeouttrees@n{0}%

```

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

```

1467 \loop
1468 \ifnum\forestdebug@typeouttrees@n<\forest@node@maxid
1469 \edef\forestdebug@typeouttrees@n{\number\numexpr\forestdebug@typeouttrees@n+1}%
1470 \ifcsdef{forestdebug@typeouttree@used@\forestdebug@typeouttrees@n}{}%
1471 \forest@for node of \forestdebug@typeouttrees@n{%

```

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

```

1472 \begingroup
1473 \forestdebug@typeouttrees@findroot
1474 \expandafter\endgroup
1475 \expandafter\edef\expandafter\forest@cn\expandafter{\forest@cn}%
1476 \forestdebug@typeouttree@build
1477 \appto\forest@temp@message{\space}%
1478 }%
1479 }%
1480 \repeat
1481 \expandafter\endgroup
1482 \expandafter\edef\expandafter#1\expandafter{\forest@temp@message}%
1483 }
1484 \def\forestdebug@typeouttrees@findroot{%
1485 \let\forestdebug@typeouttrees@next\relax
1486 \edef\forestdebug@typeouttrees@parent{\forestOve{\forest@cn}{@parent}}%
1487 \ifnum\forestdebug@typeouttrees@parent=0
1488 \else
1489 \ifcsdef{forestdebug@typeouttree@used@\forest@cn}{}%
1490 \csdef{forestdebug@typeouttree@used@\forest@cn}{}%
1491 \edef\forest@cn{\forestdebug@typeouttrees@parent}%
1492 \let\forestdebug@typeouttrees@next\forestdebug@typeouttrees@findroot
1493 }%
1494 \fi
1495 \forestdebug@typeouttrees@next
1496 }
1497 \def\forestdebug@typeouttree#1#2{% #1=root id, #2=cs to receive result
1498 \begingroup

```

```

1499 \edef\forest@temp@message{}%
1500 \forest@fornode{#1}{\forestdebug@typeouttree@build}%
1501 \expandafter\endgroup
1502 \expandafter\edef\expandafter#2\expandafter{\forest@temp@message}%
1503 }
1504 \NewDocumentCommand\forestdebugtypeouttree{o m}{%
1505 \forestdebug@typeouttree{#1}\forest@temp
1506 \typeout{\IfValueTF{#1}{#1: }{}\forest@temp}%
1507 }

```

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

```

1508 \def\forestdebug@typeouttree@build{%
1509 \eappto\forest@temp@message{\forest@cn}%
1510 \ifcsdef{forestdebug@typeouttree@used@\forest@cn}{*}{}%
1511 }%
1512 \ifcsdef{forestdebug@typeouttree@used@\forest@cn}{}{%
1513 \csdef{forestdebug@typeouttree@used@\forest@cn}{}%
1514 \forest@node@foreachchild{\forestdebug@typeouttree@build}%
1515 }%
1516 \eappto\forest@temp@message{%[
1517 ]}%
1518 }

```

6.3 Node options

6.3.1 Option-declaration mechanism

Common code for declaring options.

```

1519 \def\forest@declarehandler#1#2#3{%#1=handler for specific type,#2=option name,#3=default value
1520 \pgfkeyssetvalue{/forest/#2}{#3}%
1521 \appto\forest@node@init{\forest@init{#2}}%
1522 \pgfkeyssetvalue{/forest/#2/node@or@reg}{\forest@cn}%
1523 \forest@convert@others@to@underscores{#2}\forest@pgfmathoptionname
1524 \edef\forest@marshal{%
1525 \noexpand#1{/forest/#2}{/forest}{#2}{\forest@pgfmathoptionname}%
1526 }\forest@marshal
1527 }
1528 \def\forest@def@with@pgfeov#1#2{% \pgfeov mustn't occur in the arg of the .code handler!!!
1529 \long\def#1##1\pgfeov{#2}%
1530 }

```

Option-declaration handlers.

```

1531 \def\forest@declaretokens@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmathname
1532 \forest@declaretokens@handler@A{#1}{#2}{#3}{#4}{}%
1533 }
1534 \def\forest@declarekeylist@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmathname
1535 \forest@declaretokens@handler@A{#1}{#2}{#3}{#4}{,}%
1536 \forest@copycommandkey{#1}{#1'}%
1537 \pgfkeyssetvalue{#1'/option@name}{#3}%
1538 \forest@copycommandkey{#1+}{#1}%
1539 \pgfkeysalso{#1-/ .code={%
1540 \forest@fornode{\forest@setter@node}{%
1541 \forest@node@removekeysfromkeylist{##1}{#3}%
1542 }}}}
1543 \pgfkeyssetvalue{#1-/option@name}{#3}%
1544 }
1545 \def\forest@declaretokens@handler@A#1#2#3#4#5{% #1=key,#2=path,#3=name,#4=pgfmathname,#5=infix
1546 \pgfkeysalso{%
1547 #1/.code={\forest@set{\forest@setter@node}{#3}{##1}},
1548 #2/if #3/.code n args={3}{%
1549 \forest@toget{#3}\forest@temp@option@value

```

```

1550 \edef\forest@temp@compared@value{\unexpanded{##1}}%
1551 \ifx\forest@temp@option@value\forest@temp@compared@value
1552 \pgfkeysalso{##2}%
1553 \else
1554 \pgfkeysalso{##3}%
1555 \fi
1556 },
1557 #2/if in #3/.code n args={3}{%
1558 \forestoget{#3}\forest@temp@option@value
1559 \edef\forest@temp@compared@value{\unexpanded{##1}}%
1560 \expandafter\expandafter\expandafter\pgfutil@in@\expandafter\expandafter\expandafter{\expandafter\forest
1561 \ifpgfutil@in@
1562 \pgfkeysalso{##2}%
1563 \else
1564 \pgfkeysalso{##3}%
1565 \fi
1566 },
1567 #2/where #3/.style n args={3}{for tree={#2/if #3={##1}{##2}{##3}}},
1568 #2/where in #3/.style n args={3}{for tree={#2/if in #3={##1}{##2}{##3}}}
1569 }%
1570 \ifstrempy{#5}{%
1571 \pgfkeysalso{%
1572 #1+/.code={\forest0appto{\forest@setter@node}{#3}{#5##1}},
1573 #2/+##3/.code={\forest0preto{\forest@setter@node}{#3}{##1#5}},
1574 }%
1575 }{%
1576 \pgfkeysalso{%
1577 #1+/.code={%
1578 \forest0get{\forest@setter@node}{#3}\forest@temp
1579 \ifdefempty{\forest@temp}{%
1580 \forest0set{\forest@setter@node}{#3}{##1}%
1581 }{%
1582 \forest0appto{\forest@setter@node}{#3}{#5##1}%
1583 }%
1584 },
1585 #2/+##3/.code={%
1586 \forest0get{\forest@setter@node}{#3}\forest@temp
1587 \ifdefempty{\forest@temp}{%
1588 \forest0set{\forest@setter@node}{#3}{##1}%
1589 }{%
1590 \forest0preto{\forest@setter@node}{#3}{##1#5}%
1591 }%
1592 }%
1593 }%
1594 }%
1595 \pgfkeyssetvalue{#1/option@name}{#3}%
1596 \pgfkeyssetvalue{#1+/option@name}{#3}%
1597 \pgfkeyssetvalue{#2/+##3/option@name}{#3}%
1598 \pgfmathdeclarefunction{#4}{1}{\forest@pgfmathhelper@attribute@toks{##1}{#3}}%
1599 }
1600 \def\forest@declareautowrappedtoks@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmathname,#5=infix
1601 \forest@declaretoks@handler{#1}{#2}{#3}{#4}%
1602 \forest@copycommandkey{#1}{#1'}%
1603 \pgfkeysalso{#1/.style={#1'/.wrap value={##1}}}%
1604 \pgfkeyssetvalue{#1'/option@name}{#3}%
1605 \forest@copycommandkey{#1+}{#1+'}%
1606 \pgfkeysalso{#1+/.style={#1+'/.wrap value={##1}}}%
1607 \pgfkeyssetvalue{#1+'/option@name}{#3}%
1608 \forest@copycommandkey{#2/+##3}{#2/+##3'}%
1609 \pgfkeysalso{#2/+##3/.style={#2/+##3'/.wrap value={##1}}}%
1610 \pgfkeyssetvalue{#2/+##3'/option@name}{#3}%

```



```

1611 }
1612 \def\forest@declarereadonlydimen@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmathname
1613 % this is to have 'pt' with the correct category code
1614 \pgfutil@tempdima=\pgfkeysvalueof{/forest/#3}\relax
1615 \edef\forest@marshal{%
1616 \noexpand\pgfkeyssetvalue{/forest/#3}{\the\pgfutil@tempdima}%
1617 }\forest@marshal
1618 \pgfkeysalso{%
1619 #2/if #3/.code n args={3}{%
1620 \foresttoget{#3}\forest@temp@option@value
1621 \ifdim\forest@temp@option@value=##1\relax
1622 \pgfkeysalso{##2}%
1623 \else
1624 \pgfkeysalso{##3}%
1625 \fi
1626 },
1627 #2/where #3/.style n args={3}{for tree={#2/if #3={##1}{##2}{##3}}},
1628 }%
1629 \pgfmathdeclarefunction{#4}{1}{\forest@pgfmathhelper@attribute@dimen{##1}{#3}}%
1630 }
1631 \def\forest@declaredimen@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmathname
1632 \forest@declarereadonlydimen@handler{#1}{#2}{#3}{#4}%
1633 \pgfkeysalso{%
1634 #1/.code={%
1635 \pgfmathsetlengthmacro\forest@temp{##1}%
1636 \forestOlet{\forest@setter@node}{#3}\forest@temp
1637 },
1638 #1+/.code={%
1639 \pgfmathsetlengthmacro\forest@temp{##1}%
1640 \pgfutil@tempdima=\forestove{#3}
1641 \advance\pgfutil@tempdima\forest@temp\relax
1642 \forestOset{\forest@setter@node}{#3}{\the\pgfutil@tempdima}%
1643 },
1644 #1-/.code={%
1645 \pgfmathsetlengthmacro\forest@temp{##1}%
1646 \pgfutil@tempdima=\forestove{#3}
1647 \advance\pgfutil@tempdima-\forest@temp\relax
1648 \forestOset{\forest@setter@node}{#3}{\the\pgfutil@tempdima}%
1649 },
1650 #1*/.style={%
1651 #1={#4()}*(##1)}%
1652 },
1653 #1:/ .style={%
1654 #1={#4()/ (##1)}%
1655 },
1656 #1'/.code={%
1657 \pgfutil@tempdima=##1\relax
1658 \forestOset{\forest@setter@node}{#3}{\the\pgfutil@tempdima}%
1659 },
1660 #1'+/.code={%
1661 \pgfutil@tempdima=\forestove{#3}\relax
1662 \advance\pgfutil@tempdima##1\relax
1663 \forestOset{\forest@setter@node}{#3}{\the\pgfutil@tempdima}%
1664 },
1665 #1'-/.code={%
1666 \pgfutil@tempdima=\forestove{#3}\relax
1667 \advance\pgfutil@tempdima-##1\relax
1668 \forestOset{\forest@setter@node}{#3}{\the\pgfutil@tempdima}%
1669 },
1670 #1'*/.style={%
1671 \pgfutil@tempdima=\forestove{#3}\relax

```

```

1672     \multiply\pgfutil@tempdima##1\relax
1673     \forestOset{\forest@setter@node}{#3}{\the\pgfutil@tempdima}%
1674 },
1675 #1'/.style={%
1676     \pgfutil@tempdima=\forestove{#3}\relax
1677     \divide\pgfutil@tempdima##1\relax
1678     \forestOset{\forest@setter@node}{#3}{\the\pgfutil@tempdima}%
1679 },
1680 }%
1681 \pgfkeyssetvalue{#1/option@name}{#3}%
1682 \pgfkeyssetvalue{#1+/option@name}{#3}%
1683 \pgfkeyssetvalue{#1-/option@name}{#3}%
1684 \pgfkeyssetvalue{#1*/option@name}{#3}%
1685 \pgfkeyssetvalue{#1:/option@name}{#3}%
1686 \pgfkeyssetvalue{#1'/option@name}{#3}%
1687 \pgfkeyssetvalue{#1'+/option@name}{#3}%
1688 \pgfkeyssetvalue{#1'-/option@name}{#3}%
1689 \pgfkeyssetvalue{#1*/option@name}{#3}%
1690 \pgfkeyssetvalue{#1'/option@name}{#3}%
1691 }
1692 \def\forest@declarereadonlycount@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmathname
1693     \pgfkeysalso{
1694         #2/if #3/.code n args={3}{%
1695             \forestoget{#3}\forest@temp@option@value
1696             \ifnum\forest@temp@option@value=##1\relax
1697                 \pgfkeysalso{##2}%
1698             \else
1699                 \pgfkeysalso{##3}%
1700             \fi
1701         },
1702         #2/where #3/.style n args={3}{for tree={#2/if #3={##1}{##2}{##3}}},
1703     }%
1704     \pgfmathdeclarefunction{#4}{1}{\forest@pgfmathhelper@attribute@count{##1}{#3}}%
1705 }
1706 \def\forest@declarecount@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmathname
1707     \forest@declarereadonlycount@handler{#1}{#2}{#3}{#4}%
1708     \pgfkeysalso{
1709         #1/.code={%
1710             \pgfmathtruncatemacro\forest@temp{##1}%
1711             \forestOlet{\forest@setter@node}{#3}\forest@temp
1712         },
1713         #1+/.code={%
1714             \pgfmathtruncatemacro\forest@temp{##1}%
1715             \c@pgf@counta=\forestove{#3}\relax
1716             \advance\c@pgf@counta\forest@temp\relax
1717             \forestOset{\forest@setter@node}{#3}{\the\c@pgf@counta}%
1718         },
1719         #1-/.code={%
1720             \pgfmathtruncatemacro\forest@temp{##1}%
1721             \c@pgf@counta=\forestove{#3}\relax
1722             \advance\c@pgf@counta-\forest@temp\relax
1723             \forestOset{\forest@setter@node}{#3}{\the\c@pgf@counta}%
1724         },
1725         #1*/.code={%
1726             \pgfmathtruncatemacro\forest@temp{##1}%
1727             \c@pgf@counta=\forestove{#3}\relax
1728             \multiply\c@pgf@counta\forest@temp\relax
1729             \forestOset{\forest@setter@node}{#3}{\the\c@pgf@counta}%
1730         },
1731         #1:/.code={%
1732             \pgfmathtruncatemacro\forest@temp{##1}%

```

```

1733 \c@pgf@counta=\forestove{#3}\relax
1734 \divide\c@pgf@counta\forest@temp\relax
1735 \forestOset{\forest@setter@node}{#3}{\the\c@pgf@counta}%
1736 },
1737 #1'/.code={%
1738 \c@pgf@counta=##1\relax
1739 \forestOset{\forest@setter@node}{#3}{\the\c@pgf@counta}%
1740 },
1741 #1'+/.code={%
1742 \c@pgf@counta=\forestove{#3}\relax
1743 \advance\c@pgf@counta##1\relax
1744 \forestOset{\forest@setter@node}{#3}{\the\c@pgf@counta}%
1745 },
1746 #1'-/.code={%
1747 \c@pgf@counta=\forestove{#3}\relax
1748 \advance\c@pgf@counta-##1\relax
1749 \forestOset{\forest@setter@node}{#3}{\the\c@pgf@counta}%
1750 },
1751 #1'*/.style={%
1752 \c@pgf@counta=\forestove{#3}\relax
1753 \multiply\c@pgf@counta##1\relax
1754 \forestOset{\forest@setter@node}{#3}{\the\c@pgf@counta}%
1755 },
1756 #1':/.style={%
1757 \c@pgf@counta=\forestove{#3}\relax
1758 \divide\c@pgf@counta##1\relax
1759 \forestOset{\forest@setter@node}{#3}{\the\c@pgf@counta}%
1760 },
1761 }%
1762 \pgfkeyssetvalue{#1/option@name}{#3}%
1763 \pgfkeyssetvalue{#1+/option@name}{#3}%
1764 \pgfkeyssetvalue{#1-/option@name}{#3}%
1765 \pgfkeyssetvalue{#1*/option@name}{#3}%
1766 \pgfkeyssetvalue{#1:/option@name}{#3}%
1767 \pgfkeyssetvalue{#1'/option@name}{#3}%
1768 \pgfkeyssetvalue{#1'+/option@name}{#3}%
1769 \pgfkeyssetvalue{#1'-/option@name}{#3}%
1770 \pgfkeyssetvalue{#1'*/option@name}{#3}%
1771 \pgfkeyssetvalue{#1':/option@name}{#3}%
1772 }
1773 \def\forest@declareboolean@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmathname
1774 \pgfkeysalso{%
1775 #1/.code={%
1776 \ifstrequal{##1}{1}{%
1777 \forestOset{\forest@setter@node}{#3}{1}%
1778 }{%
1779 \ifstrequal{##1}{0}{%
1780 \forestOset{\forest@setter@node}{#3}{0}%
1781 }{%
1782 \pgfmathifthenelse{##1}{1}{0}%
1783 \forestOlet{\forest@setter@node}{#3}\pgfmathresult
1784 }%
1785 }%
1786 },
1787 #1/.default=1,
1788 #2/not #3/.code={\forestOset{\forest@setter@node}{#3}{0}},
1789 #2/if #3/.code 2 args={%
1790 \forestoget{#3}\forest@temp@option@value
1791 \ifnum\forest@temp@option@value=1
1792 \pgfkeysalso{##1}%
1793 \else

```

```

1794     \pgfkeysalso{##2}%
1795     \fi
1796 },
1797 #2/where #3/.style 2 args={for tree={#2/if #3={##1}{##2}}}
1798 }%
1799 \pgfkeyssetvalue{#1/option@name}{#3}%
1800 \pgfmathdeclarefunction{#4}{1}{\forest@pgfmathhelper@attribute@count{##1}{#3}}%
1801 }
1802 \forestset{
1803   declare toks/.code 2 args={%
1804     \forest@declarehandler\forest@declaretoks@handler{#1}{#2}%
1805   },
1806   declare autowrapped toks/.code 2 args={%
1807     \forest@declarehandler\forest@declareautowrappedtoks@handler{#1}{#2}%
1808   },
1809   declare keylist/.code 2 args={%
1810     \forest@declarehandler\forest@declarekeylist@handler{#1}{#2}%
1811   },
1812   declare readonly dimen/.code 2 args={%
1813     \pgfmathsetlengthmacro\forest@temp{#2}%
1814     \edef\forest@marshal{%
1815       \unexpanded{\forest@declarehandler\forest@declarereadonlydimen@handler{#1}}{\forest@temp}%
1816     }\forest@marshal
1817   },
1818   declare dimen/.code 2 args={%
1819     \pgfmathsetlengthmacro\forest@temp{#2}%
1820     \edef\forest@marshal{%
1821       \unexpanded{\forest@declarehandler\forest@declaredimen@handler{#1}}{\forest@temp}%
1822     }\forest@marshal
1823   },
1824   declare readonly count/.code 2 args={%
1825     \pgfmathtruncatemacro\forest@temp{#2}%
1826     \edef\forest@marshal{%
1827       \unexpanded{\forest@declarehandler\forest@declarereadonlycount@handler{#1}}{\forest@temp}%
1828     }\forest@marshal
1829   },
1830   declare count/.code 2 args={%
1831     \pgfmathtruncatemacro\forest@temp{#2}%
1832     \edef\forest@marshal{%
1833       \unexpanded{\forest@declarehandler\forest@declarecount@handler{#1}}{\forest@temp}%
1834     }\forest@marshal
1835   },
1836   declare boolean/.code 2 args={%
1837     \pgfmathtruncatemacro\forest@temp{#2}%
1838     \edef\forest@marshal{%
1839       \unexpanded{\forest@declarehandler\forest@declareboolean@handler{#1}}{\forest@temp}%
1840     }\forest@marshal
1841   },
1842   /handlers/.restore default value/.code={%
1843     \edef\forest@handlers@currentpath{\pgfkeyscurrentpath}%
1844     \pgfkeysgetvalue{\pgfkeyscurrentpath/option@name}\forest@currentoptionname
1845     \pgfkeysgetvalue{/forest/\forest@currentoptionname}\forest@temp
1846     \expandafter\pgfkeysalso\expandafter{\forest@handlers@currentpath/.expand once=\forest@temp}%
1847   },
1848   /handlers/.pgfmath/.code={%
1849     \pgfmathparse{#1}%
1850     \pgfkeysalso{\pgfkeyscurrentpath/.expand once=\pgfmathresult}%
1851   },
1852   /handlers/.wrap value/.code={%
1853     \edef\forest@handlers@wrap@currentpath{\pgfkeyscurrentpath}%
1854     \pgfkeysgetvalue{\forest@handlers@wrap@currentpath/option@name}\forest@currentoptionname

```

```

1855 \forestOget{\pgfkeysvalueof{/forest/\forest@currentoptionname/node@or@reg}}{\forest@currentoptionname}\fo
1856 \forest@def@with@pgfeov\forest@wrap@code{#1}%
1857 \expandafter\edef\expandafter\forest@wrapped@value\expandafter{\expandafter\expandonce\expandafter{\expan
1858 \pgfkeysalso{\forest@handlers@wrap@currentpath/.expand once=\forest@wrapped@value}%
1859 },
1860 /handlers/.option/.code={%
1861 \edef\forest@marshal{%
1862 \noexpand\pgfkeysalso{\pgfkeyscurrentpath={\forestoption{#1}}}%
1863 }\forest@marshal
1864 },
1865 /handlers/.register/.code={%
1866 \edef\forest@marshal{%
1867 \noexpand\pgfkeysalso{\pgfkeyscurrentpath={\forestregister{#1}}}%
1868 }\forest@marshal
1869 },
1870 /handlers/.wrap pgfmath arg/.code 2 args={%
1871 \pgfmathparse{#2}\let\forest@wrap@arg@i\pgfmathresult
1872 \edef\forest@wrap@args{{\expandonce\forest@wrap@arg@i}}%
1873 \def\forest@wrap@code##1{#1}%
1874 % here we don't call \forest@wrap@pgfmath@args@@@wrapandpasson, as compat-2.0.2-wrappgfmathargs changes t
1875 \expandafter\expandafter\expandafter\forest@temp@toks\expandafter\expandafter\expandafter{\expandafter\fo
1876 \expandafter\pgfkeysalso\expandafter{\expandafter\pgfkeyscurrentpath\expandafter=\expandafter{\the\forest
1877 },
1878 /handlers/.wrap 2 pgfmath args/.code n args={3}{%
1879 \pgfmathparse{#2}\let\forest@wrap@arg@i\pgfmathresult
1880 \pgfmathparse{#3}\let\forest@wrap@arg@ii\pgfmathresult
1881 \edef\forest@wrap@args{{\expandonce\forest@wrap@arg@i}{\expandonce\forest@wrap@arg@ii}}%
1882 \def\forest@wrap@code##1##2{#1}%
1883 \forest@wrap@pgfmath@args@@@wrapandpasson
1884 },
1885 /handlers/.wrap 3 pgfmath args/.code n args={4}{%
1886 \forest@wrap@n@pgfmath@args{#2}{#3}{#4}{-}{-}{-}{-}{-}{3}%
1887 \forest@wrap@n@pgfmath@do{#1}{3}},
1888 /handlers/.wrap 4 pgfmath args/.code n args={5}{%
1889 \forest@wrap@n@pgfmath@args{#2}{#3}{#4}{#5}{-}{-}{-}{-}{4}%
1890 \forest@wrap@n@pgfmath@do{#1}{4}},
1891 /handlers/.wrap 5 pgfmath args/.code n args={6}{%
1892 \forest@wrap@n@pgfmath@args{#2}{#3}{#4}{#5}{#6}{-}{-}{-}{5}%
1893 \forest@wrap@n@pgfmath@do{#1}{5}},
1894 /handlers/.wrap 6 pgfmath args/.code n args={7}{%
1895 \forest@wrap@n@pgfmath@args{#2}{#3}{#4}{#5}{#6}{#7}{-}{-}{6}%
1896 \forest@wrap@n@pgfmath@do{#1}{6}},
1897 /handlers/.wrap 7 pgfmath args/.code n args={8}{%
1898 \forest@wrap@n@pgfmath@args{#2}{#3}{#4}{#5}{#6}{#7}{#8}{-}{7}%
1899 \forest@wrap@n@pgfmath@do{#1}{7}},
1900 /handlers/.wrap 8 pgfmath args/.code n args={9}{%
1901 \forest@wrap@n@pgfmath@args{#2}{#3}{#4}{#5}{#6}{#7}{#8}{#9}{8}%
1902 \forest@wrap@n@pgfmath@do{#1}{8}},
1903 /handlers/.process args/.code={%
1904 \forest@processargs#1\forest@eov\forest@END
1905 },
1906 }
1907 \def\forest@wrap@n@pgfmath@args#1#2#3#4#5#6#7#8#9{%
1908 \pgfmathparse{#1}\let\forest@wrap@arg@i\pgfmathresult
1909 \ifnum#9>1 \pgfmathparse{#2}\let\forest@wrap@arg@ii\pgfmathresult\fi
1910 \ifnum#9>2 \pgfmathparse{#3}\let\forest@wrap@arg@iii\pgfmathresult\fi
1911 \ifnum#9>3 \pgfmathparse{#4}\let\forest@wrap@arg@iv\pgfmathresult\fi
1912 \ifnum#9>4 \pgfmathparse{#5}\let\forest@wrap@arg@v\pgfmathresult\fi
1913 \ifnum#9>5 \pgfmathparse{#6}\let\forest@wrap@arg@vi\pgfmathresult\fi
1914 \ifnum#9>6 \pgfmathparse{#7}\let\forest@wrap@arg@vii\pgfmathresult\fi
1915 \ifnum#9>7 \pgfmathparse{#8}\let\forest@wrap@arg@viii\pgfmathresult\fi

```

```

1916 \edef\forest@wrap@args{%
1917   {\expandonce\forest@wrap@arg@i}
1918   \ifnum#9>1 {\expandonce\forest@wrap@arg@ii}\fi
1919   \ifnum#9>2 {\expandonce\forest@wrap@arg@iii}\fi
1920   \ifnum#9>3 {\expandonce\forest@wrap@arg@iv}\fi
1921   \ifnum#9>4 {\expandonce\forest@wrap@arg@v}\fi
1922   \ifnum#9>5 {\expandonce\forest@wrap@arg@vi}\fi
1923   \ifnum#9>6 {\expandonce\forest@wrap@arg@vii}\fi
1924   \ifnum#9>7 {\expandonce\forest@wrap@arg@viii}\fi
1925 }%
1926 }
1927 \def\forest@wrap@n@pgfm@do#1#2{%
1928   \ifcase#2\relax
1929   \or\def\forest@wrap@code##1{#1}%
1930   \or\def\forest@wrap@code##1##2{#1}%
1931   \or\def\forest@wrap@code##1##2##3{#1}%
1932   \or\def\forest@wrap@code##1##2##3##4{#1}%
1933   \or\def\forest@wrap@code##1##2##3##4##5{#1}%
1934   \or\def\forest@wrap@code##1##2##3##4##5##6{#1}%
1935   \or\def\forest@wrap@code##1##2##3##4##5##6##7{#1}%
1936   \or\def\forest@wrap@code##1##2##3##4##5##6##7##8{#1}%
1937   \fi
1938   \forest@wrap@pgfm@args@@@wrapandpasson
1939 }

```

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

```

1940 \def\forest@wrap@pgfm@args@@@wrapandpasson{%
1941   \expandafter\expandafter\expandafter\forest@temp@toks
1942     \expandafter\expandafter\expandafter{%
1943       \expandafter\forest@wrap@code\forest@wrap@args}%
1944   \expandafter\pgfkeysalso\expandafter{%
1945     \expandafter\pgfkeyscurrentpath\expandafter=\expandafter{%
1946       \the\forest@temp@toks}}%
1947 }
1948 \newtoks\forest@processargs@result
1949 \newtoks\forest@processargs@current
1950 \newif\ifforest@processargs@append
1951 \def\forest@eov{\forest@eov}
1952 \def\forest@processargs#1#2\forest@END{% #1 = processing instructions, #2 = args
1953   \forest@processargs@result{}%
1954   \forest@processargs@current{}%
1955   \forest@processargs@appendfalse
1956   \forest@processargs@getins#1.\forest@END#2\forest@END
1957 }
1958 \def\forest@processargs@maybeappend{%
1959   \ifforest@processargs@append
1960     \eapptotoks\forest@processargs@result{{\the\forest@processargs@current}}%
1961     \forest@processargs@current{}%
1962   \fi
1963 }
1964 \def\forest@processargs@getins#1#2\forest@END#3\forest@END{% #1 = first instruction, #2 = rest of instruction
1965   \csname forest@processargs@ins@#1\endcsname#2\forest@END#3\forest@END
1966 }
1967 \csdef{forest@processargs@ins@.}\forest@END#1\forest@END{%
1968   \forest@processargs@maybeappend
1969   \forest@processargs@appremainingargs#1\forest@END
1970 }
1971 \def\forest@processargs@appremainingargs#1#2\forest@END{%
1972   \edef\forest@temp{\unexpanded{#1}}%
1973   \ifx\forest@temp\forest@eov
1974     \let\forest@processargs@next\forest@processargs@done

```

```

1975 \else
1976 \apptotoks\forest@processargs@result{{#1}}%
1977 \let\forest@processargs@next\forest@processargs@appremainingargs
1978 \fi
1979 \forest@processargs@next#2\forest@END
1980 }
1981 \def\forest@processargs@done#1\forest@END{%
1982 \pgfkeysalso{\pgfkeyscurrentpath/.expanded=\the\forest@processargs@result}%
1983 }
1984 \csdef{forest@processargs@ins@_}#1\forest@END#2#3\forest@END{% no processing
1985 % #1 = rest of ins, #2 = first arg, #3 = rest of args
1986 \forest@processargs@maybeappend
1987 \forest@processargs@current{#2}%
1988 \forest@processargs@appendtrue
1989 \forest@processargs@getins#1\forest@END#3\forest@END
1990 }
1991 \csdef{forest@processargs@ins@x}#1\forest@END#2#3\forest@END{% expand
1992 % #1 = rest of ins, #2 = first arg, #3 = rest of args
1993 \forest@processargs@maybeappend
1994 \etotoks\forest@processargs@current{#2}%
1995 \forest@processargs@appendtrue
1996 \forest@processargs@getins#1\forest@END#3\forest@END
1997 }
1998 \csdef{forest@processargs@ins@o}#1\forest@END#2#3\forest@END{% expand once
1999 % #1 = rest of ins, #2 = first arg, #3 = rest of args
2000 \forest@processargs@maybeappend
2001 \expandafter\forest@processargs@current\expandafter{#2}%
2002 \forest@processargs@appendtrue
2003 \forest@processargs@getins#1\forest@END#3\forest@END
2004 }
2005 \csdef{forest@processargs@ins@O}#1\forest@END#2#3\forest@END{% option
2006 % #1 = rest of ins, #2 = first arg, #3 = rest of args
2007 \forest@processargs@maybeappend
2008 \etotoks\forest@processargs@current{\forestoption{#2}}%
2009 \forest@processargs@appendtrue
2010 \forest@processargs@getins#1\forest@END#3\forest@END
2011 }
2012 \csdef{forest@processargs@ins@R}#1\forest@END#2#3\forest@END{% register
2013 % #1 = rest of ins, #2 = first arg, #3 = rest of args
2014 \forest@processargs@maybeappend
2015 \etotoks\forest@processargs@current{\forestregister{#2}}%
2016 \forest@processargs@appendtrue
2017 \forest@processargs@getins#1\forest@END#3\forest@END
2018 }
2019 \csdef{forest@processargs@ins@P}#1\forest@END#2#3\forest@END{% pgfmath expression
2020 % #1 = rest of ins, #2 = first arg, #3 = rest of args
2021 \forest@processargs@maybeappend
2022 \pgfmathparse{#2}%
2023 \expandafter\forest@processargs@current\expandafter{\pgfmathresult}%
2024 \forest@processargs@appendtrue
2025 \forest@processargs@getins#1\forest@END#3\forest@END
2026 }
2027 \csdef{forest@processargs@ins@+}#1\forest@END#2\forest@END{% join processors
2028 \forest@processargs@appendfalse
2029 \edef\forest@marshal{%
2030 \unexpanded{\forest@processargs@getins#1\forest@END}\the\forest@processargs@current}\unexpanded{#2\forest@processargs@getins#1\forest@END}
2031 }\forest@marshal
2032 }
2033 \csdef{forest@processargs@ins@r}#1\forest@END#2#3\forest@END{% reverse keylist
2034 % #1 = rest of ins, #2 = first arg, #3 = rest of args
2035 \forest@processargs@maybeappend

```

```

2036 \forest@processargs@current{}%
2037 \pgfqkeys{/forest}{split={#2}{,}{reverse@keylist}}%
2038 \forest@processargs@appendtrue
2039 \forest@processargs@getins#1\forest@END#3\forest@END
2040 }
2041 \forestset{%
2042   reverse@keylist/.code={%
2043     \epretotoks\forest@processargs@current{#1},}%
2044   },
2045 }

```

6.3.2 Registers

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

```

2046 \def\forest@pgfmathhelper@register@toks#1#2{% #1 is discarded: it is present only for analogy with options
2047   \forest@get{#2}\pgfmathresult
2048 }
2049 \def\forest@pgfmathhelper@register@dimen#1#2{%
2050   \forest@get{#2}\forest@temp
2051   \pgfmathparse{+\forest@temp}%
2052 }
2053 \def\forest@pgfmathhelper@register@count#1#2{%
2054   \forest@get{#2}\forest@temp
2055   \pgfmathtruncatemacro\pgfmathresult{\forest@temp}%
2056 }
2057 \def\forest@declareregisterhandler#1#2{%#1=handler for specific type,#2=option name
2058   \pgfkeyssetvalue{/forest/#2/node@or@reg}{register}%
2059   \forest@convert@others@to@underscores{#2}\forest@pgfmathoptionname
2060   \edef\forest@marshal{%
2061     \noexpand#1{/forest/#2}{/forest}{#2}{\forest@pgfmathoptionname}%
2062   }\forest@marshal
2063 }
2064 \def\forest@declaretoksregister@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmathname
2065   \forest@declaretoksregister@handler@A{#1}{#2}{#3}{#4}{}%
2066 }
2067 \def\forest@declarekeylistregister@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmathname
2068   \forest@declaretoksregister@handler@A{#1}{#2}{#3}{#4}{,}%
2069   \forest@copycommandkey{#1}{#1'}%
2070   \pgfkeyssetvalue{#1'/option@name}{#3}%
2071   \forest@copycommandkey{#1+}{#1}%
2072   \pgfkeysalso{#1-/.code={%
2073     \forest@fornode{register}%
2074     \forest@node@removekeysfromkeylist{##1}{#3}%
2075   }}}%
2076   \pgfkeyssetvalue{#1-/option@name}{#3}%
2077 }
2078 \def\forest@declaretoksregister@handler@A#1#2#3#4#5{% #1=key,#2=path,#3=name,#4=pgfmathname,#5=infix
2079   \pgfkeysalso{%
2080     #1/.code={\forest@set{#3}{##1}},
2081     #2/if #3/.code n args={3}{%
2082       \forest@get{#3}\forest@temp@option@value
2083       \edef\forest@temp@compared@value{\unexpanded{##1}}%
2084       \ifx\forest@temp@option@value\forest@temp@compared@value
2085         \pgfkeysalso{##2}%
2086       \else
2087         \pgfkeysalso{##3}%
2088       \fi
2089     },
2090     #2/if in #3/.code n args={3}{%
2091       \forest@get{#3}\forest@temp@option@value

```



```

2092 \edef\forest@temp@compared@value{\unexpanded{##1}}%
2093 \expandafter\expandafter\expandafter\pgfutil@in@\expandafter\expandafter\expandafter{\expandafter\forest@
2094 \ifpgfutil@in@
2095 \pgfkeysalso{##2}%
2096 \else
2097 \pgfkeysalso{##3}%
2098 \fi
2099 },
2100 }%
2101 \ifstrempy{#5}{%
2102 \pgfkeysalso{%
2103 #1+/.code={\forestrappto{#3}{#5##1}},
2104 #2/+##3/.code={\forestrpreto{#3}{##1#5}},
2105 }%
2106 }{%
2107 \pgfkeysalso{%
2108 #1+/.code={%
2109 \forestrget{#3}\forest@temp
2110 \ifdefempty{\forest@temp}{%
2111 \forestrset{#3}{##1}%
2112 }{%
2113 \forestrappto{#3}{#5##1}%
2114 }%
2115 },
2116 #2/+##3/.code={%
2117 \forestrget{#3}\forest@temp
2118 \ifdefempty{\forest@temp}{%
2119 \forestrset{#3}{##1}%
2120 }{%
2121 \forestrpreto{#3}{##1#5}%
2122 }%
2123 }%
2124 }%
2125 }%
2126 \pgfkeyssetvalue{#1/option@name}{#3}%
2127 \pgfkeyssetvalue{#1+/option@name}{#3}%
2128 \pgfkeyssetvalue{#2/+##3/option@name}{#3}%
2129 \pgfmathdeclarefunction{#4}{1}{\forest@pgfmathhelper@register@toks{##1}{#3}}%
2130 }
2131 \def\forest@declareautowrappedtoksregister@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmathname,#5=infix
2132 \forest@declaretoksregister@handler{#1}{#2}{#3}{#4}%
2133 \forest@copycommandkey{#1}{#1'}%
2134 \pgfkeysalso{#1/.style={#1'/.wrap value={##1}}}%
2135 \pgfkeyssetvalue{#1'/option@name}{#3}%
2136 \forest@copycommandkey{#1+}{#1+'}%
2137 \pgfkeysalso{#1+/.style={#1+'/.wrap value={##1}}}%
2138 \pgfkeyssetvalue{#1+'/option@name}{#3}%
2139 \forest@copycommandkey{#2/+##3}{#2/+##3'}%
2140 \pgfkeysalso{#2/+##3/.style={#2/+##3'/.wrap value={##1}}}%
2141 \pgfkeyssetvalue{#2/+##3'/option@name}{#3}%
2142 }
2143 \def\forest@declarereadonlydimenregister@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmathname
2144 \pgfkeysalso{%
2145 #2/if #3/.code n args={3}{%
2146 \forestrget{#3}\forest@temp@option@value
2147 \ifdim\forest@temp@option@value=##1\relax
2148 \pgfkeysalso{##2}%
2149 \else
2150 \pgfkeysalso{##3}%
2151 \fi
2152 },

```

```

2153 }%
2154 \pgfmathdeclarefunction{#4}{1}{\forest@pgfmathhelper@register@dimen{##1}{#3}}%
2155 }
2156 \def\forest@declaredimenregister@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmathname
2157 \forest@declaredimenregister@handler{#1}{#2}{#3}{#4}%
2158 \pgfkeysalso{%
2159   #1/.code={%
2160     \pgfmathsetlengthmacro\forest@temp{##1}%
2161     \forestrllet{#3}\forest@temp
2162   },
2163   #1+/.code={%
2164     \pgfmathsetlengthmacro\forest@temp{##1}%
2165     \pgfutil@tempdima=\forestrive{#3}
2166     \advance\pgfutil@tempdima\forest@temp\relax
2167     \forestreset{#3}{\the\pgfutil@tempdima}%
2168   },
2169   #1-/.code={%
2170     \pgfmathsetlengthmacro\forest@temp{##1}%
2171     \pgfutil@tempdima=\forestrive{#3}
2172     \advance\pgfutil@tempdima-\forest@temp\relax
2173     \forestreset{#3}{\the\pgfutil@tempdima}%
2174   },
2175   #1*/.style={%
2176     #1={#4()}*(##1)}%
2177   },
2178   #1:/ .style={%
2179     #1={#4()/(##1)}%
2180   },
2181   #1'/.code={%
2182     \pgfutil@tempdima=##1\relax
2183     \forestreset{#3}{\the\pgfutil@tempdima}%
2184   },
2185   #1'+/.code={%
2186     \pgfutil@tempdima=\forestrive{#3}\relax
2187     \advance\pgfutil@tempdima##1\relax
2188     \forestreset{#3}{\the\pgfutil@tempdima}%
2189   },
2190   #1'-/.code={%
2191     \pgfutil@tempdima=\forestrive{#3}\relax
2192     \advance\pgfutil@tempdima-##1\relax
2193     \forestreset{#3}{\the\pgfutil@tempdima}%
2194   },
2195   #1'*/.style={%
2196     \pgfutil@tempdima=\forestrive{#3}\relax
2197     \multiply\pgfutil@tempdima##1\relax
2198     \forestreset{#3}{\the\pgfutil@tempdima}%
2199   },
2200   #1':/.style={%
2201     \pgfutil@tempdima=\forestrive{#3}\relax
2202     \divide\pgfutil@tempdima##1\relax
2203     \forestreset{#3}{\the\pgfutil@tempdima}%
2204   },
2205 }%
2206 \pgfkeyssetvalue{#1/option@name}{#3}%
2207 \pgfkeyssetvalue{#1+/option@name}{#3}%
2208 \pgfkeyssetvalue{#1-/option@name}{#3}%
2209 \pgfkeyssetvalue{#1*/option@name}{#3}%
2210 \pgfkeyssetvalue{#1:/option@name}{#3}%
2211 \pgfkeyssetvalue{#1'/option@name}{#3}%
2212 \pgfkeyssetvalue{#1'+/option@name}{#3}%
2213 \pgfkeyssetvalue{#1'-/option@name}{#3}%

```

```

2214 \pgfkeyssetvalue{#1'*/option@name}{#3}%
2215 \pgfkeyssetvalue{#1':/option@name}{#3}%
2216 }
2217 \def\forest@declarereadonlycountregister@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmathname
2218 \pgfkeysalso{
2219 #2/if #3/.code n args={3}{%
2220 \forestrget{#3}\forest@temp@option@value
2221 \ifnum\forest@temp@option@value=##1\relax
2222 \pgfkeysalso{##2}%
2223 \else
2224 \pgfkeysalso{##3}%
2225 \fi
2226 },
2227 }%
2228 \pgfmathdeclarefunction{#4}{1}{\forest@pgfmathhelper@register@count{##1}{#3}}%
2229 }
2230 \def\forest@declarecountregister@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmathname
2231 \forest@declarereadonlycountregister@handler{#1}{#2}{#3}{#4}%
2232 \pgfkeysalso{
2233 #1/.code={%
2234 \pgfmathtruncatemacro\forest@temp{##1}%
2235 \forestrlet{#3}\forest@temp
2236 },
2237 #1+/.code={%
2238 \pgfmathtruncatemacro\forest@temp{##1}%
2239 \c@pgf@counta=\forestrve{#3}\relax
2240 \advance\c@pgf@counta\forest@temp\relax
2241 \forestreset{#3}{\the\c@pgf@counta}%
2242 },
2243 #1-/.code={%
2244 \pgfmathtruncatemacro\forest@temp{##1}%
2245 \c@pgf@counta=\forestrve{#3}\relax
2246 \advance\c@pgf@counta-\forest@temp\relax
2247 \forestreset{#3}{\the\c@pgf@counta}%
2248 },
2249 #1*/.code={%
2250 \pgfmathtruncatemacro\forest@temp{##1}%
2251 \c@pgf@counta=\forestrve{#3}\relax
2252 \multiply\c@pgf@counta\forest@temp\relax
2253 \forestreset{#3}{\the\c@pgf@counta}%
2254 },
2255 #1:/.code={%
2256 \pgfmathtruncatemacro\forest@temp{##1}%
2257 \c@pgf@counta=\forestrve{#3}\relax
2258 \divide\c@pgf@counta\forest@temp\relax
2259 \forestreset{#3}{\the\c@pgf@counta}%
2260 },
2261 #1'/.code={%
2262 \c@pgf@counta=##1\relax
2263 \forestreset{#3}{\the\c@pgf@counta}%
2264 },
2265 #1'+/.code={%
2266 \c@pgf@counta=\forestrve{#3}\relax
2267 \advance\c@pgf@counta##1\relax
2268 \forestreset{#3}{\the\c@pgf@counta}%
2269 },
2270 #1'-/.code={%
2271 \c@pgf@counta=\forestrve{#3}\relax
2272 \advance\c@pgf@counta-##1\relax
2273 \forestreset{#3}{\the\c@pgf@counta}%
2274 },

```

```

2275 #1'*/.style={%
2276 \c@pgf@counta=\forestrve{#3}\relax
2277 \multiply\c@pgf@counta##1\relax
2278 \forestreset{#3}{\the\c@pgf@counta}%
2279 },
2280 #1':/.style={%
2281 \c@pgf@counta=\forestrve{#3}\relax
2282 \divide\c@pgf@counta##1\relax
2283 \forestreset{#3}{\the\c@pgf@counta}%
2284 },
2285 }%
2286 \pgfkeyssetvalue{#1/option@name}{#3}%
2287 \pgfkeyssetvalue{#1+/option@name}{#3}%
2288 \pgfkeyssetvalue{#1-/option@name}{#3}%
2289 \pgfkeyssetvalue{#1*/option@name}{#3}%
2290 \pgfkeyssetvalue{#1:/option@name}{#3}%
2291 \pgfkeyssetvalue{#1'/option@name}{#3}%
2292 \pgfkeyssetvalue{#1'+/option@name}{#3}%
2293 \pgfkeyssetvalue{#1'-/option@name}{#3}%
2294 \pgfkeyssetvalue{#1'*-/option@name}{#3}%
2295 \pgfkeyssetvalue{#1':-/option@name}{#3}%
2296 }
2297 \def\forest@declarebooleanregister@handler#1#2#3#4{% #1=key,#2=path,#3=name,#4=pgfmathname
2298 \pgfkeysalso{%
2299 #1/.code={%
2300 \ifstrequal{##1}{1}{%
2301 \forestreset{#3}{1}%
2302 }{%
2303 \ifstrequal{##1}{0}{%
2304 \forestreset{#3}{0}%
2305 }{%
2306 \pgfmathifthenelse{##1}{1}{0}%
2307 \forestrlt{#3}\pgfmathresult
2308 }%
2309 }%
2310 },
2311 #1/.default=1,
2312 #2/not #3/.code={\forestreset{#3}{0}},
2313 #2/if #3/.code 2 args={%
2314 \forestrget{#3}\forest@temp@option@value
2315 \ifnum\forest@temp@option@value=1
2316 \pgfkeysalso{##1}%
2317 \else
2318 \pgfkeysalso{##2}%
2319 \fi
2320 },
2321 }%
2322 \pgfkeyssetvalue{#1/option@name}{#3}%
2323 \pgfmathdeclarefunction{#4}{1}{\forest@pgfmathhelper@register@count{##1}{#3}}%
2324 }
2325 \forestset{
2326 declare toks register/.code={%
2327 \forest@declareregisterhandler\forest@declaretoksregister@handler{#1}%
2328 \forestset{#1={}}%
2329 },
2330 declare autowrapped toks register/.code={%
2331 \forest@declareregisterhandler\forest@declareautowrappedtoksregister@handler{#1}%
2332 \forestset{#1={}}%
2333 },
2334 declare keylist register/.code={%
2335 \forest@declareregisterhandler\forest@declarekeylistregister@handler{#1}%

```

```

2336     \forestset{#1'={}}%
2337 },
2338 declare dimen register/.code={%
2339     \forest@declarerregisterhandler\forest@declaredimenregister@handler{#1}%
2340     \forestset{#1'=0pt}%
2341 },
2342 declare count register/.code={%
2343     \forest@declarerregisterhandler\forest@declarecountregister@handler{#1}%
2344     \forestset{#1'=0}%
2345 },
2346 declare boolean register/.code={%
2347     \forest@declarerregisterhandler\forest@declarebooleanregister@handler{#1}%
2348     \forestset{#1=0}%
2349 },
2350 }

```

Declare some temporary registers.

```

2351 \forestset{
2352   declare toks register=temptoksa,temptoksa={},
2353   declare toks register=temptoksb,temptoksb={},
2354   declare toks register=temptoksc,temptoksc={},
2355   declare toks register=temptoksd,temptoksd={},
2356   declare keylist register=tempkeylista,tempkeylista'={},
2357   declare keylist register=tempkeylistb,tempkeylistb'={},
2358   declare keylist register=tempkeylistc,tempkeylistc'={},
2359   declare keylist register=tempkeylistd,tempkeylistd'={},
2360   declare dimen register=tempdima,tempdima'={0pt},
2361   declare dimen register=tempdimb,tempdimb'={0pt},
2362   declare dimen register=tempdimc,tempdimc'={0pt},
2363   declare dimen register=tempdimd,tempdimd'={0pt},
2364   declare dimen register=tempdimx,tempdimx'={0pt},
2365   declare dimen register=tempdimxa,tempdimxa'={0pt},
2366   declare dimen register=tempdimxb,tempdimxb'={0pt},
2367   declare dimen register=tempdimy,tempdimy'={0pt},
2368   declare dimen register=tempdimya,tempdimya'={0pt},
2369   declare dimen register=tempdimyb,tempdimyb'={0pt},
2370   declare dimen register=tempdiml,tempdiml'={0pt},
2371   declare dimen register=tempdimla,tempdimla'={0pt},
2372   declare dimen register=tempdimlb,tempdimlb'={0pt},
2373   declare dimen register=tempdims,tempdims'={0pt},
2374   declare dimen register=tempdimsa,tempdimsa'={0pt},
2375   declare dimen register=tempdimsb,tempdimsb'={0pt},
2376   declare count register=tempcounta,tempcounta'={0},
2377   declare count register=tempcountb,tempcountb'={0},
2378   declare count register=tempcountc,tempcountc'={0},
2379   declare count register=tempcountd,tempcountd'={0},
2380   declare boolean register=tempboola,tempboola={0},
2381   declare boolean register=tempboolb,tempboolb={0},
2382   declare boolean register=tempboolc,tempboolc={0},
2383   declare boolean register=tempboold,tempboold={0},
2384 }

```

6.3.3 Declaring options

```

2385 \def\forest@node@Nametoid#1{% #1 = name
2386   \csname forest@id@of@#1\endcsname
2387 }
2388 \def\forest@node@ifnamedefined#1#2#3{% #1 = name, #2=true,#3=false
2389   \ifcsvoid{forest@id@of@#1}{#3}{#2}%
2390 }
2391 \def\forest@node@setname#1{%

```

```

2392 \def\forest@temp@setname{y}%
2393 \def\forest@temp@silent{n}%
2394 \def\forest@temp@propagating{n}%
2395 \forest@node@setnameoralias{#1}%
2396 }
2397 \def\forest@node@setname@silent#1{%
2398 \def\forest@temp@setname{y}%
2399 \def\forest@temp@silent{y}%
2400 \def\forest@temp@propagating{n}%
2401 \forest@node@setnameoralias{#1}%
2402 }
2403 \def\forest@node@setalias#1{%
2404 \def\forest@temp@setname{n}%
2405 \def\forest@temp@silent{n}%
2406 \def\forest@temp@propagating{n}%
2407 \forest@node@setnameoralias{#1}%
2408 }
2409 \def\forest@node@setalias@silent#1{%
2410 \def\forest@temp@setname{n}%
2411 \def\forest@temp@silent{y}%
2412 \def\forest@temp@propagating{n}%
2413 \forest@node@setnameoralias{#1}%
2414 }
2415 \def\forest@node@setnameoralias#1{%
2416 \ifstrempy{#1}{%
2417 \forest@node@setnameoralias{node@\forest@cn}%
2418 }{%
2419 \forest@node@ifnamedefined{#1}{%
2420 \if y\forest@temp@propagating
2421 % this will find a unique name, eventually:
2422 \@escapeif{\forest@node@setnameoralias{#1@\forest@cn}}%
2423 \else\@escapeif{%
2424 \if y\forest@temp@setname
2425 \edef\forest@marshal{%
2426 \ifstrequal{\forest@name}{#1}%
2427 }\forest@marshal{%
2428 % same name, no problem
2429 }{%
2430 \@escapeif{\forest@node@setnameoralias@nameclash{#1}}%
2431 }%
2432 \else\@escapeif{% setting an alias: clashing with alias is not a problem
2433 \forest@get{\forest@node@Nametoid{#1}}{name}\forest@temp
2434 \expandafter\ifstrequal\expandafter{\forest@temp}{#1}{%
2435 \forest@node@setnameoralias@nameclash{#1}%
2436 }{%
2437 \forest@node@setnameoralias@do{#1}%
2438 }%
2439 }\fi
2440 }\fi
2441 }{%
2442 \forest@node@setnameoralias@do{#1}%
2443 }%
2444 }%
2445 }
2446 \def\forest@node@setnameoralias@nameclash#1{%
2447 \if y\forest@temp@silent
2448 \forest@for node{\forest@node@Nametoid{#1}}{%
2449 \def\forest@temp@propagating{y}%
2450 \forest@node@setnameoralias{}%
2451 }%
2452 \forest@node@setnameoralias@do{#1}%

```

```

2453 \else
2454   \PackageError{forest}{Node name "#1" is already used}{}%
2455 \fi
2456 }
2457 \def\forest@node@setnameoralias@do#1{%
2458   \if y\forest@temp@setname
2459     \csdef{forest@id@of@\forest@ve{name}}{}%
2460     \forest@eset{name}{#1}%
2461   \fi
2462   \csedef{forest@id@of#1}{\forest@cn}%
2463 }
2464 \forestset{
2465   TeX/.code={#1},
2466   TeX'/.code={\appto\forest@externalize@loadimages{#1}#1},
2467   TeX''/.code={\appto\forest@externalize@loadimages{#1}},
2468   options/.code={\forestset{#1}},
2469   typeout/.style={TeX={\typeout{#1}}},
2470   declare toks={name}{},
2471   name/.code={% override the default setter
2472     \forest@fornode{\forest@setter@node}{\forest@node@setname{#1}}%
2473   },
2474   name/.default={},
2475   name'/.code={% override the default setter
2476     \forest@fornode{\forest@setter@node}{\forest@node@setname@silent{#1}}%
2477   },
2478   name'/.default={},
2479   alias/.code={\forest@fornode{\forest@setter@node}{\forest@node@setalias{#1}}},
2480   alias'/.code={\forest@fornode{\forest@setter@node}{\forest@node@setalias@silent{#1}}},
2481   begin draw/.code={\begin{tikzpicture}},
2482   end draw/.code={\end{tikzpicture}},
2483   declare keylist register=default preamble,
2484   default preamble'={},
2485   declare keylist register=preamble,
2486   preamble'={},
2487   declare autowrapped toks={content}{},
2488   % #1 = which option to split, #2 = separator (one char!), #3 = receiving options
2489   %% begin listing region: split_option
2490   split option/.style n args=3{split/.process args={0}{#1}{#2}{#3}}
2491   %% end listing region: split_option
2492   ,
2493   split register/.style n args=3{% #1 = which register to split, #2 = separator (one char!), #3 = receiving options
2494     split/.process args={R}{#1}{#2}{#3},
2495   },
2496   TeX={%
2497     \def\forest@split@sourcevalues{}%
2498     \def\forest@split@sourcevalue{}%
2499     \def\forest@split@receivingoptions{}%
2500     \def\forest@split@receivingoption{}%
2501   },
2502   split/.code n args=3{% #1 = string to split, #2 = separator (one char!), #3 = receiving options
2503     \forest@saveandrestoremacro\forest@split@sourcevalues{%
2504       \forest@saveandrestoremacro\forest@split@sourcevalue{%
2505         \forest@saveandrestoremacro\forest@split@receivingoptions{%
2506           \forest@saveandrestoremacro\forest@split@receivingoption{%
2507             \def\forest@split@sourcevalues{#1#2}%
2508             \edef\forest@split@receivingoptions{#3}%
2509             \def\forest@split@receivingoption{}%
2510             \safeloop
2511             \expandafter\forest@split\expandafter{\forest@split@sourcevalues}{#2}\forest@split@sourcevalue\
2512             \ifdefempty\forest@split@receivingoptions{}{%
2513               \expandafter\forest@split\expandafter{\forest@split@receivingoptions}{,}\forest@temp\forest@s

```

```

2514         \ifdefempty\forest@temp{}\let\forest@split@receivingoption\forest@temp\def\forest@temp{}\}%
2515     }%
2516     \edef\forest@marshal{%
2517         \noexpand\pgfkeysalso{\forest@split@receivingoption={\expandonce{\forest@split@sourcevalue}}}%
2518     }\forest@marshal
2519     \ifdefempty\forest@split@sourcevalues{\forest@tempfalse}\forest@temptrue}%
2520     \ifforest@temp
2521     \saferepeat
2522     }}}}
2523 },
2524 declare count={grow}{270},
2525 TeX={% a hack for grow-reversed connection, and compass-based grow specification
2526     \forest@copycommandkey{/forest/grow}{/forest/grow@@}%
2527     %\pgfkeysgetvalue{/forest/grow/.cmd}\forest@temp
2528     %\pgfkeyslet{/forest/grow@@/.cmd}\forest@temp
2529 },
2530 grow/.style={grow@={#1},reversed=0},
2531 grow'/.style={grow@={#1},reversed=1},
2532 grow''/.style={grow@={#1}},
2533 grow@/.is choice,
2534 grow@/east/.style={/forest/grow@@=0},
2535 grow@/north east/.style={/forest/grow@@=45},
2536 grow@/north/.style={/forest/grow@@=90},
2537 grow@/north west/.style={/forest/grow@@=135},
2538 grow@/west/.style={/forest/grow@@=180},
2539 grow@/south west/.style={/forest/grow@@=225},
2540 grow@/south/.style={/forest/grow@@=270},
2541 grow@/south east/.style={/forest/grow@@=315},
2542 grow@/.unknown/.code={\let\forest@temp@grow\pgfkeyscurrentname
2543     \pgfkeysalso{/forest/grow@@/.expand once=\forest@temp@grow}},
2544 declare boolean={reversed}{0},
2545 declare toks={parent anchor}{},
2546 declare toks={child anchor}{},
2547 declare toks={anchor}{base},
2548 Autoforward={anchor}{
2549     node options--anchor,
2550     node options+= {anchor={#1}}
2551 },
2552 anchor'/.style={anchor@no@compass=true,anchor=#1},
2553 anchor+/.style={anchor@no@compass=true,anchor+=#1},
2554 anchor-/.style={anchor@no@compass=true,anchor-=#1},
2555 anchor*/.style={anchor@no@compass=true,anchor*=#1},
2556 anchor:/.style={anchor@no@compass=true,anchor:=#1},
2557 anchor'+/.style={anchor@no@compass=true,anchor'+=#1},
2558 anchor'-/.style={anchor@no@compass=true,anchor'-=#1},
2559 anchor'*/.style={anchor@no@compass=true,anchor'*=#1},
2560 anchor':/.style={anchor@no@compass=true,anchor':=#1},
2561 % /tikz/forest anchor/.style={
2562 %     /forest/TeX={\forestanchortotikzanchor{#1}\forest@temp@anchor},
2563 %     anchor/.expand once=\forest@temp@anchor
2564 % },
2565 declare toks={calign}{midpoint},
2566 TeX={%
2567     \forest@copycommandkey{/forest/calign}{/forest/calign'}%
2568 },
2569 calign/.is choice,
2570 calign/child/.style={calign'=child},
2571 calign/first/.style={calign'=child,calign primary child=1},
2572 calign/last/.style={calign'=child,calign primary child=-1},
2573 calign with current/.style={for parent/.wrap pgfmath arg={calign=child,calign primary child=##1}{n}},
2574 calign with current edge/.style={for parent/.wrap pgfmath arg={calign=child edge,calign primary child=##1}{

```



```

2575 calign/child edge/.style={calign'=child edge},
2576 calign/midpoint/.style={calign'=midpoint},
2577 calign/center/.style={calign'=midpoint,calign primary child=1,calign secondary child=-1},
2578 calign/edge midpoint/.style={calign'=edge midpoint},
2579 calign/fixed angles/.style={calign'=fixed angles},
2580 calign/fixed edge angles/.style={calign'=fixed edge angles},
2581 calign/.unknown/.code={\PackageError{forest}{unknown calign '\pgfkeyscurrentname'}{}}},
2582 declare count={calign primary child}{1},
2583 declare count={calign secondary child}{-1},
2584 declare count={calign primary angle}{-35},
2585 declare count={calign secondary angle}{35},
2586 calign child/.style={calign primary child={#1}},
2587 calign angle/.style={calign primary angle={-#1},calign secondary angle={#1}},
2588 declare toks={tier}{},
2589 declare toks={fit}{tight},
2590 declare boolean={ignore}{0},
2591 declare boolean={ignore edge}{0},
2592 no edge/.style={edge'={},ignore edge},
2593 declare keylist={edge}{draw},
2594 declare toks={edge path}{%
2595   \noexpand\path[\forestoption{edge}]%
2596   (\forestOve{\forestove{@parent}}{name}.parent anchor)--(\forestove{name}.child anchor)
2597   % =
2598   % (!u.parent anchor)--(.child anchor)\forestoption{edge label};
2599   \forestoption{edge label}};
2600 },
2601 edge path'/.style={
2602   edge path={%
2603     \noexpand\path[\forestoption{edge}]%
2604     #1%
2605     \forestoption{edge label};
2606   }
2607 },
2608 declare toks={edge label}{},
2609 declare boolean={phantom}{0},
2610 baseline/.style={alias={forest@baseline@node}},
2611 declare readonly count={id}{0},
2612 declare readonly count={n}{0},
2613 declare readonly count={n'}{0},
2614 declare readonly count={n children}{-1},
2615 declare readonly count={level}{-1},
2616 declare dimen=x{0pt},
2617 declare dimen=y{0pt},
2618 declare dimen={s}{0pt},
2619 declare dimen={l}{6ex}, % just in case: should be set by the calibration
2620 declare dimen={s sep}{0.6666em},
2621 declare dimen={l sep}{1ex}, % just in case: calibration!
2622 declare keylist={node options}{anchor=base},
2623 declare toks={tikz}{},
2624 afterthought/.style={tikz+=#{#1}},
2625 label/.style={tikz+={\path[late options={%
2626   name=\forestoption{name},label={#1}}];}},
2627 pin/.style={tikz+={\path[late options={%
2628   name=\forestoption{name},pin={#1}}];}},
2629 declare toks={content format}{\forestoption{content}},
2630 plain content/.style={content format={\forestoption{content}}},
2631 math content/.style={content format={\noexpand\ensuremath{\forestoption{content}}}},
2632 declare toks={node format}{%
2633   \noexpand\node
2634   (\forestoption{name})%
2635   [\forestoption{node options}]}

```

```

2636     {\foresteoption{content format}};%
2637 },
2638 node format'/.style={
2639     node format={\noexpand\node{\foresteoption{name}}#1;}
2640 },
2641 tabular@environment/.style={content format={%
2642     \noexpand\begin{tabular}[\foresteoption{base}]{\foresteoption{align}}%
2643     \foresteoption{content}%
2644     \noexpand\end{tabular}%
2645 }},
2646 declare toks={align}{},
2647 TeX={%
2648     \forest@copycommandkey{/forest/align}{/forest/align'}%
2649     %\pgfkeysgetvalue{/forest/align/.@cmd}\forest@temp
2650     %\pgfkeyslet{/forest/align'/.@cmd}\forest@temp
2651 },
2652 align/.is choice,
2653 align/.unknown/.code={%
2654     \edef\forest@marshal{%
2655         \noexpand\pgfkeysalso{%
2656             align'={\pgfkeyscurrentname},%
2657             tabular@environment
2658         }%
2659     }\forest@marshal
2660 },
2661 align/center/.style={align'={@{}c@{}}},tabular@environment},
2662 align/left/.style={align'={@{}l@{}}},tabular@environment},
2663 align/right/.style={align'={@{}r@{}}},tabular@environment},
2664 declare toks={base}{t},
2665 TeX={%
2666     \forest@copycommandkey{/forest/base}{/forest/base'}%
2667     %\pgfkeysgetvalue{/forest/base/.@cmd}\forest@temp
2668     %\pgfkeyslet{/forest/base'/.@cmd}\forest@temp
2669 },
2670 base/.is choice,
2671 base/top/.style={base'=t},
2672 base/bottom/.style={base'=b},
2673 base/.unknown/.style={base'/.expand once=\pgfkeyscurrentname},
2674 unknown to/.store in=\forest@unknownto,
2675 unknown to=node options,
2676 unknown key error/.code={\PackageError{forest}{Unknown keyval: \detokenize{#1}}{}},
2677 content to/.store in=\forest@contentto,
2678 content to=content,
2679 .unknown/.code={%
2680     \expandafter\pgfutil@in@\expandafter.\expandafter{\pgfkeyscurrentname}%
2681     \ifpgfutil@in@
2682         \expandafter\forest@relatednode@option@setter\pgfkeyscurrentname=#1\forest@END
2683     \else
2684         \edef\forest@marshal{%
2685             \noexpand\pgfkeysalso{\forest@unknownto={\pgfkeyscurrentname=\unexpanded{#1}}}%
2686         }\forest@marshal
2687     \fi
2688 },
2689 get node boundary/.code={%
2690     \forest@get{boundary}\forest@node@boundary
2691     \def#1{%
2692         \forest@extendpath#1\forest@node@boundary{\pgfpoint{\forestove{x}}{\forestove{y}}}%
2693     },
2694 % get min 1 tree boundary/.code={%
2695 %     \forest@get{tree@boundary}{negative}{\the\numexpr\forestove{grow}-90\relax}#1},
2696 % get max 1 tree boundary/.code={%

```

```

2697 % \forest@get@tree@boundary{positive}{\the\numexpr\forestove{grow}-90\relax}#1},
2698 get min s tree boundary/.code={%
2699 \forest@get@tree@boundary{negative}{\forestove{grow}}#1},
2700 get max s tree boundary/.code={%
2701 \forest@get@tree@boundary{positive}{\forestove{grow}}#1},
2702 use as bounding box/.style={%
2703 before drawing tree={
2704 tikz+/.expanded={%
2705 \noexpand\pgfresetboundingbox
2706 \noexpand\useasboundingbox
2707 ($(.anchor)+(\forestoption{min x},\forestoption{min y}))$)
2708 rectangle
2709 ($(.anchor)+(\forestoption{max x},\forestoption{max y}))$)
2710 ;
2711 }
2712 }
2713 },
2714 use as bounding box'/.style={%
2715 before drawing tree={
2716 tikz+/.expanded={%
2717 \noexpand\pgfresetboundingbox
2718 \noexpand\useasboundingbox
2719 ($(.anchor)+(\forestoption{min x}+\pgfkeysvalueof{/pgf/outer xsep}/2+\pgfkeysvalueof{/pgf/inner xsep}))$)
2720 rectangle
2721 ($(.anchor)+(\forestoption{max x}-\pgfkeysvalueof{/pgf/outer xsep}/2-\pgfkeysvalueof{/pgf/inner xsep}))$)
2722 ;
2723 }
2724 }
2725 },
2726 }%
2727 \def\forest@iftikzkey#1#2#3{% #1 = key name, #2 = true code, #3 = false code
2728 \forest@temptrue
2729 \pgfkeysifdefined{/tikz/\pgfkeyscurrentname}{}{%
2730 \pgfkeysifdefined{/tikz/\pgfkeyscurrentname/.@cmd}{}{%
2731 \pgfkeysifdefined{/pgf/\pgfkeyscurrentname}{}{%
2732 \pgfkeysifdefined{/pgf/\pgfkeyscurrentname/.@cmd}{}{%
2733 \forest@tempfalse
2734 }}}}%
2735 \ifforest@temp\@escapeif{#2}\else\@escapeif{#3}\fi
2736 }
2737 \def\forest@ifoptionortikzkey#1#2#3{% #1 = key name, #2 = true code, #3 = false code
2738 \forest@temptrue
2739 \pgfkeysifdefined{/forest/\pgfkeyscurrentname}{}{%
2740 \pgfkeysifdefined{/forest/\pgfkeyscurrentname/.@cmd}{}{%
2741 \forest@iftikzkey{#1}{}{}%
2742 }}}%
2743 \ifforest@temp\@escapeif{#2}\else\@escapeif{#3}\fi
2744 }
2745 \def\forest@get@tree@boundary#1#2#3{%#1=pos/neg,#2=grow,#3=receiving cs
2746 \def#3{}%
2747 \forest@node@getedge{#1}{#2}\forest@temp@boundary
2748 \forest@extendpath#3\forest@temp@boundary{\pgfpoint{\forestove{x}}{\forestove{y}}}%
2749 }
2750 \def\forest@setter@node{\forest@cn}%
2751 \def\forest@relatednode@option@compat@ignoreinvalidsteps#1{#1}
2752 \def\forest@relatednode@option@setter#1.#2=#3\forest@END{%
2753 \forest@forthis{%
2754 \forest@relatednode@option@compat@ignoreinvalidsteps{%
2755 \forest@nameandgo{#1}%
2756 \let\forest@setter@node\forest@cn
2757 }%

```

```

2758 }%
2759 \ifnum\forest@setter@node=0
2760 \else
2761   \forestset{#2={#3}}%
2762 \fi
2763 \def\forest@setter@node{\forest@cn}%
2764 }%
2765 \def\forest@split#1#2#3#4{% #1=list (assuming that the list is nonempty and finishes with the separator), #2
2766   \def\forest@split@@##1#2##2\forest@split@@##3##4{\def##3{##1}\def##4{##2}}%
2767   \forest@split@@#1\forest@split@@{#3}{#4}}

```

6.3.4 Option propagation

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

```

2768 \forestset{
2769   for tree'/.style 2 args={#1,for children={for tree'={#1}{#2}},#2},
2770   if/.code n args={3}{%
2771     \pgfmathparse{#1}%
2772     \ifnum\pgfmathresult=0
2773       \escapeif{\pgfkeysalso{#3}}%
2774     \else
2775       \escapeif{\pgfkeysalso{#2}}%
2776     \fi
2777   },
2778   if nodewalk valid/.code n args={3}{%
2779     \edef\forest@marshal{%
2780       \unexpanded{\forest@forthis{%
2781         \forest@nodewalk{%
2782           on invalid={fake}{#1},
2783           TeX={\global\let\forest@global@temp\forest@cn}
2784         }}%
2785       }%
2786       \def\forest@cn{\forest@cn}\unexpanded{%
2787         \ifnum\forest@global@temp=0
2788           \escapeif{\pgfkeysalso{#3}}%
2789         \else
2790           \escapeif{\pgfkeysalso{#2}}%
2791         \fi}%
2792     }\forest@marshal
2793   },
2794   if nodewalk empty/.code n args={3}{%
2795     \forest@forthis{%
2796       \forest@nodewalk{%
2797         on invalid={fake}{#1},
2798         TeX={\global\let\forest@global@temp\forest@nodewalk@n},
2799       }}%
2800     }%
2801     \ifnum\forest@global@temp=0
2802       \escapeif{\pgfkeysalso{#2}}%
2803     \else
2804       \escapeif{\pgfkeysalso{#3}}%
2805     \fi
2806   },
2807   where/.style n args={3}{for tree={if={#1}{#2}{#3}}},
2808   where nodewalk valid/.style n args={3}{for tree={if nodewalk valid={#1}{#2}{#3}}},
2809   where nodewalk empty/.style n args={3}{for tree={if nodewalk empty={#1}{#2}{#3}}},
2810   repeat/.code 2 args={%
2811     \pgfmathtruncatemacro\forest@temp{#1}%
2812     \expandafter\forest@repeatkey\expandafter{\forest@temp}{#2}%
2813   },
2814   until/.code 2 args={%

```

```

2815 \ifstrempy{#1}{%
2816 \forest@untilkey{\ifnum\forest@cn=0\else\relax\forestloopbreak\fi}{on invalid={fake}{#2}}%
2817 }{%
2818 \forest@untilkey{\pgfmathifthenelse{#1}{\noexpand\forestloopbreak}{""}\pgfmathresult}{#2}%
2819 }%
2820 },
2821 while/.code 2 args={%
2822 \ifstrempy{#1}{%
2823 \forest@untilkey{\ifnum\forest@cn=0\relax\forestloopbreak\fi}{on invalid={fake}{#2}}%
2824 }{%
2825 \forest@untilkey{\pgfmathifthenelse{not(#1)}{\noexpand\forestloopbreak}{""}\pgfmathresult}{#2}%
2826 }%
2827 },
2828 do until/.code 2 args={%
2829 \ifstrempy{#1}{%
2830 \forest@dountilkey{\ifnum\forest@cn=0\else\relax\forestloopbreak\fi}{on invalid={fake}{#2}}%
2831 }{%
2832 \forest@dountilkey{\pgfmathifthenelse{#1}{\noexpand\forestloopbreak}{""}\pgfmathresult}{#2}%
2833 }%
2834 },
2835 do while/.code 2 args={%
2836 \ifstrempy{#1}{%
2837 \forest@dountilkey{\ifnum\forest@cn=0\relax\forestloopbreak\fi}{on invalid={fake}{#2}}%
2838 }{%
2839 \forest@dountilkey{\pgfmathifthenelse{not(#1)}{\noexpand\forestloopbreak}{""}\pgfmathresult}{#2}%
2840 }%
2841 },
2842 until nodewalk valid/.code 2 args={%
2843 \forest@untilkey{\forest@forthis{%
2844 \forest@nodewalk{on invalid={fake}{#1},TeX={\ifnum\forest@cn=0\relax\else\forestloopbreak\fi}}}{#2}}%
2845 },
2846 while nodewalk valid/.code 2 args={%
2847 \forest@untilkey{\forest@forthis{%
2848 \forest@nodewalk{on invalid={fake}{#1},TeX={\ifnum\forest@cn=0\relax\forestloopbreak\fi}}}{#2}}%
2849 },
2850 do until nodewalk valid/.code 2 args={%
2851 \forest@dountilkey{\forest@forthis{%
2852 \forest@nodewalk{on invalid={fake}{#1},TeX={\ifnum\forest@cn=0\relax\else\forestloopbreak\fi}}}{#2}}%
2853 },
2854 do while nodewalk valid/.code 2 args={%
2855 \forest@dountilkey{\forest@forthis{%
2856 \forest@nodewalk{on invalid={fake}{#1},TeX={\ifnum\forest@cn=0\relax\forestloopbreak\fi}}}{#2}}%
2857 },
2858 until nodewalk empty/.code 2 args={%
2859 \forest@untilkey{\forest@forthis{%
2860 \forest@nodewalk{on invalid={fake}{#1},TeX={\ifnum\forest@nodewalk@n=0\relax\forestloopbreak\fi}}}{#2}}%
2861 },
2862 while nodewalk empty/.code 2 args={%
2863 \forest@untilkey{\forest@forthis{%
2864 \forest@nodewalk{on invalid={fake}{#1},TeX={\ifnum\forest@nodewalk@n=0\relax\else\forestloopbreak\fi}}}{#2}}%
2865 },
2866 do until nodewalk empty/.code 2 args={%
2867 \forest@dountilkey{\forest@forthis{%
2868 \forest@nodewalk{on invalid={fake}{#1},TeX={\ifnum\forest@nodewalk@n=0\relax\forestloopbreak\fi}}}{#2}}%
2869 },
2870 do while nodewalk empty/.code 2 args={%
2871 \forest@dountilkey{\forest@forthis{%
2872 \forest@nodewalk{on invalid={fake}{#1},TeX={\ifnum\forest@nodewalk@n=0\relax\else\forestloopbreak\fi}}}{#2}}%
2873 },
2874 break/.code={\forestloopBreak{#1}},
2875 break/.default=0,

```

```

2876 }
2877 \def\forest@repeatkey#1#2{%
2878   \safeRKloop
2879   \ifnum\safeRKloopn>#1\relax
2880   \csuse{safeRKbreak@\the\safeRKloop@depth true}%
2881   \fi
2882   \expandafter\unless\csname ifsafeRKbreak@\the\safeRKloop@depth\endcsname
2883   \pgfkeysalso{#2}%
2884   \safeRKrepeat
2885 }
2886 \def\forest@untilkey#1#2{% #1 = condition, #2 = keys
2887   \safeRKloop
2888   #1%
2889   \expandafter\unless\csname ifsafeRKbreak@\the\safeRKloop@depth\endcsname
2890   \pgfkeysalso{#2}%
2891   \safeRKrepeat
2892 }
2893 \def\forest@dountilkey#1#2{% #1 = condition, #2 = keys
2894   \safeRKloop
2895   \pgfkeysalso{#2}%
2896   #1%
2897   \expandafter\unless\csname ifsafeRKbreak@\the\safeRKloop@depth\endcsname
2898   \safeRKrepeat
2899 }
2900 \def\forestloopbreak{%
2901   \csname safeRKbreak@\the\safeRKloop@depth true\endcsname
2902 }
2903 \def\forestloopBreak#1{%
2904   \csname safeRKbreak@\number\numexpr\the\safeRKloop@depth-#1\relax true\endcsname
2905 }
2906 \def\forestloopcount{%
2907   \csname safeRKloopn@\number\numexpr\the\safeRKloop@depth\endcsname
2908 }
2909 \def\forestloopCount#1{%
2910   \csname safeRKloopn@\number\numexpr\the\safeRKloop@depth-#1\endcsname
2911 }
2912 \pgfmathdeclarefunction{forestloopcount}{1}{%
2913   \edef\pgfmathresult{\forestloopCount{\ifstrepty{#1}{0}{#1}}}%
2914 }
2915 \forest@copycommandkey{/forest/repeat}{/forest/nodewalk/repeat}
2916 \forest@copycommandkey{/forest/while}{/forest/nodewalk/while}
2917 \forest@copycommandkey{/forest/do while}{/forest/nodewalk/do while}
2918 \forest@copycommandkey{/forest/until}{/forest/nodewalk/until}
2919 \forest@copycommandkey{/forest/do until}{/forest/nodewalk/do until}
2920 \forest@copycommandkey{/forest/if}{/forest/nodewalk/if}
2921 \forest@copycommandkey{/forest/if nodewalk valid}{/forest/nodewalk/if nodewalk valid}
2922 %

```

6.4 Aggregate functions

```

2923 \forestset{
2924   aggregate postparse/.is choice,
2925   aggregate postparse/int/.code={%
2926     \let\forest@aggregate@pgfmathpostparse\forest@aggregate@pgfmathpostparse@toint},
2927   aggregate postparse/none/.code={%
2928     \let\forest@aggregate@pgfmathpostparse\relax},
2929   aggregate postparse/print/.code={%
2930     \let\forest@aggregate@pgfmathpostparse\forest@aggregate@pgfmathpostparse@print},
2931   aggregate postparse/macro/.code={%
2932     \let\forest@aggregate@pgfmathpostparse\forest@aggregate@pgfmathpostparse@usemacro},

```

```

2933 aggregate postparse macro/.store in=\forest@aggregate@pgfmthpostparse@macro,
2934 }
2935 \def\forest@aggregate@pgfmthpostparse#print{%
2936   \pgfmthprintnumberof{\pgfmthresult}{\pgfmthresult}%
2937 }
2938 \def\forest@aggregate@pgfmthpostparse@toint{%
2939   \expandafter\forest@split\expandafter{\pgfmthresult.}{.}\pgfmthresult\forest@temp
2940 }
2941 \def\forest@aggregate@pgfmthpostparse@usemacro{%
2942   \forest@aggregate@pgfmthpostparse@macro
2943 }
2944 \let\forest@aggregate@pgfmthpostparse\relax
2945 \pgfkeys{
2946   /handlers/.aggregate/.code n args=4{%
2947     % #1 = start value
2948     % #2 = pgfmth expression for every step
2949     % #3 = pgfmth expression for after walk
2950     % #4 = nodewalk
2951     \edef\forest@marshal{%
2952       \unexpanded{\forest@aggregate{#1}{#2}{#3}{#4}}{\pgfkeyscurrentpath}%
2953     }\forest@marshal
2954   },
2955   /handlers/.sum/.style 2 args={/handlers/.aggregate={0}{(##1)+(##1)}{##1}{#2}},
2956   /handlers/.count/.style={/handlers/.aggregate={0}{1+(##1)}{##1}{#1}},
2957   /handlers/.average/.style 2 args={/handlers/.aggregate={0}{(##1)+(##1)}{##1}{\forestregister{aggregate n}}{#2}},
2958   /handlers/.product/.style 2 args={/handlers/.aggregate={1}{(##1)*(##1)}{##1}{#2}},
2959   /handlers/.min/.style 2 args={/handlers/.aggregate={}{\min(##1,##1)}{##1}{#2}},
2960   /handlers/.max/.style 2 args={/handlers/.aggregate={}{\max(##1,##1)}{##1}{#2}},
2961   /forest/declare count register={aggregate n},
2962 }
2963
2964 \def\forest@aggregate#1#2#3#4#5{%
2965   % #5 = current path
2966   \def\forest@aggregate@result{#1}%
2967   \forest@forthis{%
2968     \forestreset{aggregate n}{0}%
2969     \forest@nodewalk{#4}{%
2970       TeX={%
2971         \forestreset{aggregate n}{\number\numexpr\forestrv{aggregate n}+1}%
2972         \def\forest@marshal##1{\pgfmthparse{#2}}%
2973         \expandafter\forest@marshal\expandafter{\forest@aggregate@result}%
2974         \let\forest@aggregate@result\pgfmthresult
2975       }%
2976     }%
2977   }%
2978   \def\forest@marshal##1{\pgfmthparse{#3}}%
2979   \expandafter\forest@marshal\expandafter{\forest@aggregate@result}%
2980   \let\forest@aggregate@result\pgfmthresult
2981   \let\forest@temp@pgfmthpostparse\pgfmthpostparse
2982   \let\pgfmthpostparse\forest@aggregate@pgfmthpostparse
2983   \pgfmthqparse{\forest@aggregate@result pt}%
2984   \let\pgfmthpostparse\forest@temp@pgfmthpostparse
2985   \let\forest@aggregate@result\pgfmthresult
2986   \pgfkeysalso{#5/.expand once=\forest@aggregate@result}%
2987 }

```

6.4.1 pgfmth extensions

```

2988 \pgfmthdeclarefunction{strequal}{2}{%
2989   \ifstrequal{#1}{#2}{\def\pgfmthresult{1}}{\def\pgfmthresult{0}}%
2990 }
2991 \pgfmthdeclarefunction{instr}{2}{%

```

```

2992 \pgfutil@in@{#1}{#2}%
2993 \ifpgfutil@in@def\pgfmathresult{1}\else\def\pgfmathresult{0}\fi
2994 }
2995 \pgfmathdeclarefunction{strcat}{...}{%
2996 \edef\pgfmathresult{\forest@strip@braces{#1}}%
2997 }
2998 \pgfmathdeclarefunction{min_s}{2}{% #1 = node, #2 = context node (for growth rotation)
2999 \forest@forthis{%
3000 \forest@nameandgo{#1}%
3001 \forest@compute@minmax@ls{#2}%
3002 \pgfmathsetmacro\pgfmathresult{\forestove{min@s}}%
3003 }%
3004 }
3005 \pgfmathdeclarefunction{min_l}{2}{% #1 = node, #2 = context node (for growth rotation)
3006 \forest@forthis{%
3007 \forest@nameandgo{#1}%
3008 \forest@compute@minmax@ls{#2}%
3009 \pgfmathsetmacro\pgfmathresult{\forestove{min@l}}%
3010 }%
3011 }
3012 \pgfmathdeclarefunction{max_s}{2}{% #1 = node, #2 = context node (for growth rotation)
3013 \forest@forthis{%
3014 \forest@nameandgo{#1}%
3015 \forest@compute@minmax@ls{#2}%
3016 \pgfmathsetmacro\pgfmathresult{\forestove{max@s}}%
3017 }%
3018 }
3019 \pgfmathdeclarefunction{max_l}{2}{% #1 = node, #2 = context node (for growth rotation)
3020 \forest@forthis{%
3021 \forest@nameandgo{#1}%
3022 \forest@compute@minmax@ls{#2}%
3023 \pgfmathsetmacro\pgfmathresult{\forestove{max@l}}%
3024 }%
3025 }
3026 \def\forest@compute@minmax@ls#1{% #1 = nodewalk; in the context of which node?
3027 {%
3028 \pgftransformreset
3029 \forest@forthis{%
3030 \forest@nameandgo{#1}%
3031 \pgftransformrotate{-\forestove{grow}}%
3032 }%
3033 \forestoget{min x}\forest@temp@minx
3034 \forestoget{min y}\forest@temp@miny
3035 \forestoget{max x}\forest@temp@maxx
3036 \forestoget{max y}\forest@temp@maxy
3037 \pgfpointransformed{\pgfqpoint{\forest@temp@minx}{\forest@temp@miny}}%
3038 \forestoeset{min@l}{\the\pgf@x}%
3039 \forestoeset{min@s}{\the\pgf@y}%
3040 \forestoeset{max@l}{\the\pgf@x}%
3041 \forestoeset{max@s}{\the\pgf@y}%
3042 \pgfpointransformed{\pgfqpoint{\forest@temp@minx}{\forest@temp@maxy}}%
3043 \ifdim\pgf@x<\forestove{min@l}\relax\forestoeset{min@l}{\the\pgf@x}\fi
3044 \ifdim\pgf@y<\forestove{min@s}\relax\forestoeset{min@s}{\the\pgf@y}\fi
3045 \ifdim\pgf@x>\forestove{max@l}\relax\forestoeset{max@l}{\the\pgf@x}\fi
3046 \ifdim\pgf@y>\forestove{max@s}\relax\forestoeset{max@s}{\the\pgf@y}\fi
3047 \pgfpointransformed{\pgfqpoint{\forest@temp@maxx}{\forest@temp@miny}}%
3048 \ifdim\pgf@x<\forestove{min@l}\relax\forestoeset{min@l}{\the\pgf@x}\fi
3049 \ifdim\pgf@y<\forestove{min@s}\relax\forestoeset{min@s}{\the\pgf@y}\fi
3050 \ifdim\pgf@x>\forestove{max@l}\relax\forestoeset{max@l}{\the\pgf@x}\fi
3051 \ifdim\pgf@y>\forestove{max@s}\relax\forestoeset{max@s}{\the\pgf@y}\fi
3052 \pgfpointransformed{\pgfqpoint{\forest@temp@maxx}{\forest@temp@maxy}}%

```



```

3053 \ifdim\pgf@x<\forestove{min@1}\relax\forestoeset{min@1}{\the\pgf@x}\fi
3054 \ifdim\pgf@y<\forestove{min@s}\relax\forestoeset{min@s}{\the\pgf@y}\fi
3055 \ifdim\pgf@x>\forestove{max@1}\relax\forestoeset{max@1}{\the\pgf@x}\fi
3056 \ifdim\pgf@y>\forestove{max@s}\relax\forestoeset{max@s}{\the\pgf@y}\fi
3057 % smuggle out
3058 \edef\forest@marshal{%
3059 \noexpand\forestoeset{min@1}{\forestove{min@1}}%
3060 \noexpand\forestoeset{min@s}{\forestove{min@s}}%
3061 \noexpand\forestoeset{max@1}{\forestove{max@1}}%
3062 \noexpand\forestoeset{max@s}{\forestove{max@s}}%
3063 }\expandafter
3064 }\forest@marshal
3065 }
3066 \def\forest@pgfmathhelper@attribute@toks#1#2{%
3067 \forest@forthis{%
3068 \forest@nameandgo{#1}%
3069 \ifnum\forest@cn=0
3070 \def\pgfmathresult{%
3071 \else
3072 \forestoget{#2}\pgfmathresult
3073 \fi
3074 }%
3075 }
3076 \def\forest@pgfmathhelper@attribute@dimen#1#2{%
3077 \forest@forthis{%
3078 \forest@nameandgo{#1}%
3079 \ifnum\forest@cn=0
3080 \def\pgfmathresult{Opt}%
3081 \else
3082 \forestoget{#2}\forest@temp
3083 \pgfmathparse{+\forest@temp}%
3084 \fi
3085 }%
3086 }
3087 \def\forest@pgfmathhelper@attribute@count#1#2{%
3088 \forest@forthis{%
3089 \forest@nameandgo{#1}%
3090 \ifnum\forest@cn=0
3091 \def\pgfmathresult{0}%
3092 \else
3093 \forestoget{#2}\forest@temp
3094 \pgfmathtruncatemacro\pgfmathresult{\forest@temp}%
3095 \fi
3096 }%
3097 }
3098 \pgfmathdeclarefunction*{id}{1}{%
3099 \forest@forthis{%
3100 \forest@nameandgo{#1}%
3101 \let\pgfmathresult\forest@cn
3102 }%
3103 }

```

6.5 Nodewalk

Setup machinery.

```

3104 \def\forest@nodewalk@n{0}
3105 \def\forest@nodewalk@historyback{0,}
3106 \def\forest@nodewalk@historyforward{0,}
3107 \def\forest@nodewalk@origin{0}
3108 \def\forest@nodewalk@config@everystep@independent@before#1{% #1 = every step keylist
3109 \forestreset{every step}{#1}%

```

```

3110 }
3111 \def\forest@nodewalk@config@everystep@independent@after{%
3112   \noexpand\forest@rset{every step}{\forest@rv{every step}}%
3113 }
3114 \def\forest@nodewalk@config@history@independent@before{%
3115   \def\forest@nodewalk@n{0}%
3116   \edef\forest@nodewalk@origin{\forest@cn}%
3117   \def\forest@nodewalk@historyback{0,%}%
3118   \def\forest@nodewalk@historyforward{0,%}%
3119 }
3120 \def\forest@nodewalk@config@history@independent@after{%
3121   \edef\noexpand\forest@nodewalk@n{\expandonce{\forest@nodewalk@n}}%
3122   \edef\noexpand\forest@nodewalk@origin{\expandonce{\forest@nodewalk@origin}}%
3123   \edef\noexpand\forest@nodewalk@historyback{\expandonce{\forest@nodewalk@historyback}}%
3124   \edef\noexpand\forest@nodewalk@historyforward{\expandonce{\forest@nodewalk@historyforward}}%
3125 }
3126 \def\forest@nodewalk@config@everystep@shared@before#1{% #1 = every step keylist
3127 \def\forest@nodewalk@config@everystep@shared@after{}
3128 \def\forest@nodewalk@config@history@shared@before{}
3129 \def\forest@nodewalk@config@history@shared@after{}
3130 \def\forest@nodewalk@config@everystep@inherited@before#1{% #1 = every step keylist
3131 \let\forest@nodewalk@config@everystep@inherited@after\forest@nodewalk@config@everystep@independent@after
3132 \def\forest@nodewalk@config@history@inherited@before{}
3133 \let\forest@nodewalk@config@history@inherited@after\forest@nodewalk@config@history@independent@after
3134 \def\forest@nodewalk@#1#2{% #1 = nodewalk, #2 = every step keylist
3135   \def\forest@nodewalk@config@everystep@method{independent}%
3136   \def\forest@nodewalk@config@history@method{independent}%
3137   \def\forest@nodewalk@config@oninvalid{inherited}%
3138   \forest@saveandrestoremacro\forest@nodewalk@oninvalid{%
3139     \forest@nodewalk@#1}{#2}%
3140   }%
3141 }
3142 \def\forest@nodewalk@#1#2{% #1 = nodewalk, #2 = every step keylist
3143   \edef\forest@marshal{%
3144     \noexpand\pgfqkeys{/forest/nodewalk}{\unexpanded{#1}}%
3145     \csname forest@nodewalk@config@everystep@\forest@nodewalk@config@everystep@method @after\endcsname
3146     \csname forest@nodewalk@config@history@\forest@nodewalk@config@history@method @after\endcsname
3147   }%
3148   \csname forest@nodewalk@config@everystep@\forest@nodewalk@config@everystep@method @before\endcsname{#2}%
3149   \csname forest@nodewalk@config@history@\forest@nodewalk@config@history@method @before\endcsname
3150   \forest@nodewalk@fakefalse
3151   \forest@marshal
3152 }
3153 \pgfmathdeclarefunction{valid}{1}{%
3154   \forest@forthis{%
3155     \forest@nameandgo{#1}%
3156     \edef\pgfmathresult{\ifnum\forest@cn=0 0\else 1\fi}%
3157   }%
3158 }
3159 \pgfmathdeclarefunction{invalid}{1}{%
3160   \forest@forthis{%
3161     \forest@nameandgo{#1}%
3162     \edef\pgfmathresult{\ifnum\forest@cn=0 1\else 0\fi}%
3163   }%
3164 }
3165 \newif\ifforest@nodewalk@fake
3166 \def\forest@nodewalk@oninvalid{error}
3167 \def\forest@nodewalk@makestep{%
3168   \ifnum\forest@cn=0
3169     \csname forest@nodewalk@makestep@oninvalid@\forest@nodewalk@config@oninvalid\endcsname
3170   \else

```

```

3171 \forest@nodewalk@makestep@
3172 \fi
3173 }
3174 \def\forest@nodewalk@makestep@oninvalid@step{\forest@nodewalk@makestep@}
3175 \def\forest@nodewalk@makestep@oninvalid@error{\PackageError{forest}{nodewalk stepped to the invalid node\Mess
3176 \let\forest@nodewalk@makestep@oninvalid@fake\relax
3177 \def\forest@nodewalk@makestep@oninvalid@compatfake{%
3178 \forest@deprecated{last step in stack "\forest@nodewalk@currentstepname", which stepped on an invalid node;
3179 }%
3180 \def\forest@nodewalk@makestep@oninvalid@inherited{\csname forest@nodewalk@makestep@oninvalid@\forest@nodewalk
3181 \def\forest@nodewalk@makestep@{%
3182 \ifforest@nodewalk@fake
3183 \else
3184 \edef\forest@nodewalk@n{\number\numexpr\forest@nodewalk@n+1}%
3185 \epreto\forest@nodewalk@historyback{\forest@cn,}%
3186 \def\forest@nodewalk@historyforward{0,}%
3187 \ifforestdebugnodewalks\forest@nodewalk@makestep@debug\fi
3188 \forest@process@keylist@register{every step}%
3189 \fi
3190 }
3191 \def\forest@nodewalk@makestep@debug{%
3192 \edef\forest@marshal{%
3193 \noexpand\typeout{\ifforest@nodewalk@fake fake \fi "\forest@nodewalk@currentstepname" step to node id=\fo
3194 }\forest@marshal
3195 }%
3196 \def\forest@handlers@savecurrentpath{%
3197 \edef\pgfkeyscurrentkey{\pgfkeyscurrentpath}%
3198 \let\forest@currentkey\pgfkeyscurrentkey
3199 \pgfkeys@split@path
3200 \edef\forest@currentpath{\pgfkeyscurrentpath}%
3201 \let\forest@currentname\pgfkeyscurrentname
3202 }
3203 \pgfkeys{/handlers/save current path/.code={\forest@handlers@savecurrentpath}}
3204 \newif\ifforest@nodewalkstephandler@style
3205 \newif\ifforest@nodewalkstephandler@autostep
3206 \newif\ifforest@nodewalkstephandler@stripfakesteps
3207 \newif\ifforest@nodewalkstephandler@muststartatvalidnode
3208 \newif\ifforest@nodewalkstephandler@makefor
3209 \let\forest@nodewalkstephandler@styletrueorfalse\forest@nodewalkstephandler@stylefalse
3210 \def\forest@nodewalk@currentstepname{}
3211 \forestset{
3212 /forest/define@step/style/.is if=forest@nodewalkstephandler@style,
3213 /forest/define@step/autostep/.is if=forest@nodewalkstephandler@autostep,
3214 % the following is useful because some macros use grouping (by \forest@forthis or similar) and therefore, a
3215 /forest/define@step/strip fake steps/.is if=forest@nodewalkstephandler@stripfakesteps,
3216 % this can never happen with autosteps ...
3217 /forest/define@step/autostep/.append code={%
3218 \ifforest@nodewalkstephandler@autostep
3219 \forest@nodewalkstephandler@stripfakestepsfalse
3220 \fi
3221 },
3222 /forest/define@step/must start at valid node/.is if=forest@nodewalkstephandler@muststartatvalidnode,
3223 /forest/define@step/n args/.store in=\forest@nodewalkstephandler@nargs,
3224 /forest/define@step/make for/.is if=forest@nodewalkstephandler@makefor,
3225 /forest/define@step/@bare/.style={strip fake steps=false,must start at valid node=false,make for=false},
3226 define long step/.code n args=3{%
3227 \forest@nodewalkstephandler@styletrueorfalse % true for end users; but in the package, most of steps are
3228 \forest@nodewalkstephandler@autostepfalse
3229 \forest@nodewalkstephandler@stripfakestepstrue
3230 \forest@nodewalkstephandler@muststartatvalidnodetrue % most steps can only start at a valid node
3231 \forest@nodewalkstephandler@makefortrue % make for prefix?

```

```

3232 \def\forest@nodewalkstephandler@nargs{0}%
3233 \pgfqkeys{/forest/define@step}{#2}%
3234 \forest@temp@toks{#3}% handler code
3235 \ifforest@nodewalkstephandler@style
3236 \expandafter\forest@temp@toks\expandafter{%
3237 \expandafter\pgfkeysalso\expandafter{\the\forest@temp@toks}%
3238 }%
3239 \fi
3240 \ifforest@nodewalkstephandler@autostep
3241 \apptotoks\forest@temp@toks{\forest@nodewalk@makestep}%
3242 \fi
3243 \ifforest@nodewalkstephandler@stripfakesteps
3244 \expandafter\forest@temp@toks\expandafter{\expandafter\forest@nodewalk@stripfakesteps\expandafter{\the\
3245 \fi
3246 \ifforest@nodewalkstephandler@muststartatvalidnode
3247 \edef\forest@marshal{%
3248 \noexpand\forest@temp@toks{%
3249 \unexpanded{%
3250 \ifnum\forest@cn=0
3251 \csname forest@nodewalk@start@oninvalid\forest@nodewalk@oninvalid\endcsname{#1}%
3252 \else
3253 }%
3254 \noexpand\escapeif{\the\forest@temp@toks}%
3255 \noexpand\fi
3256 }%
3257 }\forest@marshal
3258 \fi
3259 \pretotoks\forest@temp@toks{\appto\forest@nodewalk@currentstepname{, #1}}%
3260 \expandafter\forest@temp@toks\expandafter{\expandafter\forest@saveandrestoremacro\expandafter\forest@node
3261 \ifforestdebugnodewalks
3262 \epretotoks\forest@temp@toks{\noexpand\typeout{Starting step "#1" from id=\noexpand\forest@cn
3263 \ifnum\forest@nodewalkstephandler@nargs>0 \space with args ####1\fi
3264 \ifnum\forest@nodewalkstephandler@nargs>1 ,####2\fi
3265 \ifnum\forest@nodewalkstephandler@nargs>2 ,####3\fi
3266 \ifnum\forest@nodewalkstephandler@nargs>3 ,####4\fi
3267 \ifnum\forest@nodewalkstephandler@nargs>4 ,####5\fi
3268 \ifnum\forest@nodewalkstephandler@nargs>5 ,####6\fi
3269 \ifnum\forest@nodewalkstephandler@nargs>6 ,####7\fi
3270 \ifnum\forest@nodewalkstephandler@nargs>7 ,####8\fi
3271 \ifnum\forest@nodewalkstephandler@nargs>8 ,####9\fi
3272 }}%
3273 \fi
3274 \def\forest@temp{/forest/nodewalk/#1/.code}%
3275 \ifnum\forest@nodewalkstephandler@nargs<2
3276 \eappto\forest@temp{=}%
3277 \else\ifnum\forest@nodewalkstephandler@nargs=2
3278 \eappto\forest@temp{ 2 args=}%
3279 \else
3280 \eappto\forest@temp{ n args={\forest@nodewalkstephandler@nargs}}%
3281 \fi\fi
3282 \eappto\forest@temp{{\the\forest@temp@toks}}%
3283 \expandafter\pgfkeysalso\expandafter{\forest@temp}%
3284 \ifforest@nodewalkstephandler@makefor
3285 \ifnum\forest@nodewalkstephandler@nargs=0
3286 \forestset{%
3287 for #1/.code={\forest@forstepwrapper{#1}{##1}},
3288 }%
3289 \else\ifnum\forest@nodewalkstephandler@nargs=1
3290 \forestset{%
3291 for #1/.code 2 args={\forest@forstepwrapper{#1}{##1}}{##2}},
3292 }%

```

```

3293     \else
3294     \forestset{%
3295         for #1/.code n args/.expanded=%
3296             {\number\numexpr\forest@nodewalkstephandler@nargs+1}%
3297             {\noexpand\forest@forstepwrapper{#1\ifnum\forest@nodewalkstephandler@nargs>0=\fi\forest@util@narg
3298             }%
3299     \fi\fi
3300     \fi
3301 },
3302 }
3303 \pgfqkeys{/handlers}{
3304     .nodewalk style/.code={\forest@handlers@savecurrentpath\pgfkeysalso{%
3305         \forest@currentpath/nodewalk/\forest@currentname/.style={#1}%
3306     }},
3307 }

```

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

```

3308 \def\forest@forstepwrapper#1#2{\forest@forthis{\forest@nodewalk{#1}{#2}}}
3309 \def\forest@util@nargs#1#2#3{% #1 = prefix (#, ##, ...), #2 = n args, #3=start; returns {#start}{#start+1}...
3310     \ifnum#2>0 {#1\number\numexpr#3+1}\fi
3311     \ifnum#2>1 {#1\number\numexpr#3+2}\fi
3312     \ifnum#2>2 {#1\number\numexpr#3+3}\fi
3313     \ifnum#2>3 {#1\number\numexpr#3+4}\fi
3314     \ifnum#2>4 {#1\number\numexpr#3+5}\fi
3315     \ifnum#2>5 {#1\number\numexpr#3+6}\fi
3316     \ifnum#2>6 {#1\number\numexpr#3+7}\fi
3317     \ifnum#2>7 {#1\number\numexpr#3+8}\fi
3318     \ifnum#2>8 {#1\number\numexpr#3+9}\fi
3319 }
3320 \def\forest@nodewalk@start@oninvalid@fake#1{}
3321 \def\forest@nodewalk@start@oninvalid@real#1{}
3322 \def\forest@nodewalk@start@oninvalid@error#1{\PackageError{forest}{nodewalk step "#1" cannot start at the inv

```

Define long-form single-step walks.

```

3323 \forestset{
3324     define long step={current}{autostep}{},
3325     define long step={next}{autostep}{\edef\forest@cn{\forestove{@next}}},
3326     define long step={previous}{autostep}{\edef\forest@cn{\forestove{@previous}}},
3327     define long step={parent}{autostep}{\edef\forest@cn{\forestove{@parent}}},
3328     define long step={first}{autostep}{\edef\forest@cn{\forestove{@first}}},
3329     define long step={last}{autostep}{\edef\forest@cn{\forestove{@last}}},
3330     define long step={sibling}{autostep}{%
3331         \edef\forest@cn{%
3332             \ifnum\forestove{@previous}=0
3333             \forestove{@next}%
3334         \else
3335             \forestove{@previous}%
3336         \fi
3337     }%
3338 },
3339     define long step={next node}{autostep}{\edef\forest@cn{\forest@node@linearnextid}},
3340     define long step={previous node}{autostep}{\edef\forest@cn{\forest@node@linearpreviousid}},
3341     define long step={first leaf}{autostep}{%
3342         \safeloop
3343         \edef\forest@cn{\forestove{@first}}%
3344         \unless\ifnum\forestove{@first}=0
3345             \saferepeat
3346         },
3347     define long step={first leaf'}{autostep}{%
3348         \safeloop

```

```

3349 \unless\ifnum\forestove{@first}=0
3350 \edef\forest@cn{\forestove{@first}}%
3351 \saferepeat
3352 },
3353 define long step={last leaf}{autostep}{%
3354 \safeloop
3355 \edef\forest@cn{\forestove{@last}}%
3356 \unless\ifnum\forestove{@last}=0
3357 \saferepeat
3358 },
3359 define long step={last leaf'}{autostep}{%
3360 \safeloop
3361 \unless\ifnum\forestove{@last}=0
3362 \edef\forest@cn{\forestove{@last}}%
3363 \saferepeat
3364 },
3365 define long step={next leaf}{style,strip fake steps=false}{group={do until={n_children()==0}{next node}}},
3366 define long step={previous leaf}{style,strip fake steps=false}{group={do until={n_children()==0}{previous node}}},
3367 define long step={next on tier}{autostep}{%
3368 \def\forest@temp{#1}%
3369 \ifx\forest@temp\pgfkeysnovalue@text
3370 \foresttoget{tier}\forest@nodewalk@giventier
3371 \else
3372 \def\forest@nodewalk@giventier{#1}%
3373 \fi
3374 \edef\forest@cn{\forest@node@linearnextnotdescendantid}%
3375 \safeloop
3376 \foresttoget{tier}\forest@nodewalk@tier
3377 \unless\ifx\forest@nodewalk@tier\forest@nodewalk@giventier
3378 \edef\forest@cn{\forest@node@linearnextid}%
3379 \saferepeat
3380 },
3381 define long step={previous on tier}{autostep}{%
3382 \def\forest@temp{#1}%
3383 \ifx\forest@temp\pgfkeysnovalue@text
3384 \foresttoget{tier}\forest@nodewalk@giventier
3385 \else
3386 \def\forest@nodewalk@giventier{#1}%
3387 \fi
3388 \safeloop
3389 \edef\forest@cn{\forest@node@linearpreviousid}%
3390 \foresttoget{tier}\forest@nodewalk@tier
3391 \unless\ifx\forest@nodewalk@tier\forest@nodewalk@giventier
3392 \saferepeat
3393 },
3394 %
3395 define long step={root}{autostep,must start at valid node=false}{%
3396 \edef\forest@cn{\forest@node@rootid},
3397 define long step={root'}{autostep,must start at valid node=false}{%
3398 \forest0ifdefined{\forest@root}{@parent}{\edef\forest@cn{\forest@root}}{\edef\forest@cn{0}}%
3399 },
3400 define long step={origin}{autostep,must start at valid node=false}{\edef\forest@cn{\forest@nodewalk@origin}}
3401 %
3402 define long step={n}{autostep,n args=1}{%
3403 \pgfmathtruncatemacro\forest@temp{n}{#1}%
3404 \edef\forest@cn{\forest@node@nthchildid{\forest@temp{n}}}%
3405 },
3406 define long step={n}{autostep,make for=false,n args=1}{%
3407 % Yes, twice. ;-)
3408 % n=1 and n(ext)
3409 \def\forest@nodewalk@temp{#1}%

```

```

3410 \ifx\forest@nodewalk@temp\pgfkeysnovalue@text
3411 \edef\forest@cn{\forest@next}%
3412 \else
3413 \pgfmathtruncatemacro\forest@temp{n{#1}}%
3414 \edef\forest@cn{\forest@node@nthchildid{\forest@temp}}%
3415 \fi
3416 },
3417 define long step={n'}{autostep,n args=1}{%
3418 \pgfmathtruncatemacro\forest@temp{n{#1}}%
3419 \edef\forest@cn{\forest@node@nbarthchildid{\forest@temp}}%
3420 },
3421 define long step={to tier}{autostep,n args=1}{%
3422 \def\forest@nodewalk@giventier{#1}%
3423 \safeloop
3424 \forest@toget{tier}\forest@nodewalk@tier
3425 \unless\ifx\forest@nodewalk@tier\forest@nodewalk@giventier
3426 \forest@toget{@parent}\forest@cn
3427 \saferepeat
3428 },
3429 %
3430 define long step={name}{autostep,n args=1,must start at valid node=false}{%
3431 \edef\forest@cn{%
3432 \forest@node@ifnamedefined{#1}{\forest@node@Nametoid{#1}}{0}}%
3433 }%
3434 },
3435 define long step={id}{autostep,n args=1,must start at valid node=false}{%
3436 \forest@ifdefined{#1}{@parent}{\edef\forest@cn{#1}}{\edef\forest@cn{0}}%
3437 },
3438 define long step={Nodewalk}{n args=3,@bare}{% #1 = config, #2 = nodewalk
3439 \def\forest@nodewalk@config@everystep@method{independent}%
3440 \def\forest@nodewalk@config@history@method{shared}%
3441 \def\forest@nodewalk@config@oninvalid{inherited}%
3442 \forest@saveandrestoremacro\forest@nodewalk@oninvalid{%
3443 \pgfqkeys{/forest/nodewalk@config}{#1}%
3444 \forest@Nodewalk{#2}{#3}%
3445 }%
3446 },
3447 define long step={nodewalk}{n args=2,@bare}{% #1 = nodewalk, #2 = every step
3448 \forest@nodewalk{#1}{#2}%
3449 },
3450 define long step={nodewalk'}{n args=1,@bare}{% #1 = nodewalk, #2 = every step
3451 \def\forest@nodewalk@config@everystep@method{inherited}%
3452 \def\forest@nodewalk@config@history@method{independent}%
3453 \def\forest@nodewalk@config@oninvalid{inherited}%
3454 \forest@saveandrestoremacro\forest@nodewalk@oninvalid{%
3455 \forest@Nodewalk{#1}{}%
3456 }%
3457 },
3458 % these must be defined explicitly, as prefix "for" normally introduces the every-step keylist
3459 for nodewalk/.code 2 args={%
3460 \forest@forthis{\forest@nodewalk{#1}{#2}}},
3461 for nodewalk'/.code={%
3462 \def\forest@nodewalk@config@everystep@method{inherited}%
3463 \def\forest@nodewalk@config@history@method{independent}%
3464 \def\forest@nodewalk@config@oninvalid{inherited}%
3465 \forest@saveandrestoremacro\forest@nodewalk@oninvalid{%
3466 \forest@forthis{\forest@Nodewalk{#1}{}}}%
3467 }%
3468 },
3469 for Nodewalk/.code n args=3{% #1 = config, #2 = nodewalk, #3 = every-step
3470 \def\forest@nodewalk@config@everystep@method{inherited}%

```

```

3471 \def\forest@nodewalk@config@history@method{independent}%
3472 \def\forest@nodewalk@config@oninvalid{inherited}%
3473 \forest@saveandrestoremacro\forest@nodewalk@oninvalid{%
3474 \pgfqkeys{/forest/nodewalk@config}{#1}%
3475 \forest@forthis{\forest@Nodewalk{#2}{#3}}%
3476 }%
3477 },
3478 copy command key={/forest/for nodewalk}{/forest/nodewalk/for nodewalk},
3479 copy command key={/forest/for nodewalk'}{/forest/nodewalk/for nodewalk'},
3480 copy command key={/forest/for Nodewalk}{/forest/nodewalk/for Nodewalk},
3481 declare keylist register=every step,
3482 every step'={},
3483 %%% begin nodewalk config
3484 nodewalk@config/.cd,
3485 every@step/.is choice,
3486 every@step/independent/.code={},
3487 every@step/inherited/.code={},
3488 every@step/shared/.code={},
3489 every step/.store in=\forest@nodewalk@config@everystep@method,
3490 every step/.prefix style={every@step=#1},
3491 @history/.is choice,
3492 @history/independent/.code={},
3493 @history/inherited/.code={},
3494 @history/shared/.code={},
3495 history/.store in=\forest@nodewalk@config@history@method,
3496 history/.prefix style={@history=#1},
3497 on@invalid/.is choice,
3498 on@invalid/error/.code={},
3499 on@invalid/fake/.code={},
3500 on@invalid/step/.code={},
3501 on@invalid/inherited/.code={},
3502 on invalid/.store in=\forest@nodewalk@config@oninvalid,
3503 on invalid/.prefix style={on@invalid=#1},
3504 %%% end nodewalk config
3505 }
3506 \forestset{
3507 declare toks register=branch@temp@toks,
3508 branch@temp@toks={},
3509 declare keylist register=branched@nodewalk,
3510 branched@nodewalk={},
3511 define long step={branch}{n args=1,@bare,style}{@branch={#1}{branch@build@realstep,branch@build@fakestep}},
3512 define long step={branch'}{n args=1,@bare,style}{@branch={#1}{branch@build@realstep}},
3513 @branch/.style 2 args={%
3514 save and restore register={branched@nodewalk}{
3515 branch@temp@toks={},
3516 split/.process args={r}{#1}{,}{#2},
3517 branch@temp@style/.style/.register=branch@temp@toks,
3518 branch@temp@style,
3519 branch@temp@style/.style/.register=branched@nodewalk,
3520 branch@temp@style,
3521 }
3522 },
3523 nodewalk/branch@build@realstep/.style={% #1 = nodewalk for this branch
3524 branch@temp@toks/.expanded={for nodewalk={\unexpanded{#1}}{
3525 branched@nodewalk+/.expanded={id=\noexpand\forestoption{id}},
3526 \forestregister{branch@temp@toks}}},
3527 },
3528 nodewalk/branch@build@fakestep/.style={% #1 = nodewalk for this branch
3529 branch@temp@toks/.expanded={for nodewalk={\unexpanded{#1}}{
3530 \forestregister{branch@temp@toks}}},
3531 },

```



```

3532 define long step={group}{autostep}{\forest@go{#1}},
3533 nodewalk/fake/.code={%
3534   \forest@saveandrestoreifcs{forest@nodewalk@fake}{%
3535     \forest@nodewalk@fake>true
3536     \pgfkeysalso{#1}%
3537   }%
3538 },
3539 nodewalk/real/.code={%
3540   \forest@saveandrestoreifcs{forest@nodewalk@fake}{%
3541     \forest@nodewalk@fake>false
3542     \pgfkeysalso{#1}%
3543   }%
3544 },
3545 declare keylist register=filtered@nodewalk,
3546 filtered@nodewalk={},
3547 define long step={filter}{n args=2,@bare,style}{% #1 = nodewalk, #2 = condition
3548   save and restore register={filtered@nodewalk}{
3549     filtered@nodewalk'={},
3550     Nodewalk=%
3551     {history=inherited}%
3552     {#1}%
3553     {if={#2}{filtered@nodewalk+/.expanded={id=\forestoption{id}}}{}},
3554     filtered@nodewalk@style/.style/.register=filtered@nodewalk,
3555     filtered@nodewalk@style
3556   }
3557 },
3558 on@invalid/.is choice,
3559 on@invalid/error/.code={},
3560 on@invalid/fake/.code={},
3561 on@invalid/step/.code={},
3562 on invalid/.code 2 args={%
3563   \pgfkeysalso{/forest/on@invalid={#1}}%
3564   \forest@saveandrestoremacro\forest@nodewalk@oninvalid{%
3565     \def\forest@nodewalk@oninvalid{#1}%
3566     \pgfkeysalso{#2}%
3567   }%
3568 },
3569 define long step={strip fake steps}{n args=1,@bare}{%
3570   \forest@nodewalk@stripfakesteps{\pgfkeysalso{#1}}},
3571 define long step={walk back}{n args=1,@bare}{%
3572   \pgfmathtruncatemacro\forest@temp{n{#1}}%
3573   \forest@nodewalk@walklist{\forest@nodewalk@historyforward}{\forest@nodewalk@historyback}{\ifnum\forest@cn>\forest@temp\forest@nodewalk@back@updatehistory}
3574   \forest@nodewalk@back@updatehistory
3575 },
3576 nodewalk/walk back/.default=1,
3577 define long step={jump back}{n args=1,@bare}{%
3578   \pgfmathtruncatemacro\forest@temp{n{(#1)+\ifnum\forest@cn=0 0\else1\fi}}%
3579   \forest@nodewalk@walklist{\forest@nodewalk@historyforward}{\forest@nodewalk@historyback}{\forest@temp@n-1}
3580   \forest@nodewalk@back@updatehistory
3581 },
3582 nodewalk/jump back/.default=1,
3583 define long step={back}{n args=1,@bare}{%
3584   \pgfmathtruncatemacro\forest@temp{n{#1}}%
3585   \forest@nodewalk@walklist{\forest@nodewalk@historyforward}{\forest@nodewalk@historyback}{\ifnum\forest@cn>\forest@temp\forest@nodewalk@back@updatehistory}
3586   \forest@nodewalk@back@updatehistory
3587 },
3588 nodewalk/back/.default=1,
3589 define long step={walk forward}{n args=1,@bare}{%
3590   \pgfmathtruncatemacro\forest@temp{n{#1}}%
3591   \forest@nodewalk@walklist{\forest@nodewalk@historyback}{\forest@nodewalk@historyforward}{0}{\forest@temp@n}
3592   \forest@nodewalk@forward@updatehistory

```

```

3593 },
3594 nodewalk/walk forward/.default=1,
3595 define long step={jump forward}{n args=1,@bare}{%
3596   \pgfmathtruncatemacro\forest@temp@n{#1}%
3597   \forest@nodewalk@walklist{\forest@nodewalk@historyback}{\forest@nodewalk@historyforward}{\forest@temp@n-1}
3598   \forest@nodewalk@forward@updatehistory
3599 },
3600 nodewalk/jump forward/.default=1,
3601 define long step={forward}{n args=1,@bare}{%
3602   \pgfmathtruncatemacro\forest@temp@n{#1}%
3603   \forest@nodewalk@walklist{\forest@nodewalk@historyback}{\forest@nodewalk@historyforward}{0}{\forest@temp@n}
3604   \forest@nodewalk@forward@updatehistory
3605 },
3606 nodewalk/forward/.default=1,
3607 define long step={last valid'}{@bare}{%
3608   \ifnum\forest@cn=0
3609     \forest@nodewalk@tolastvalid
3610     \forest@nodewalk@makestep
3611   \fi
3612 },
3613 define long step={last valid}{@bare}{%
3614   \forest@nodewalk@tolastvalid
3615 },
3616 define long step={reverse}{n args=1,@bare,make for}{%
3617   \forest@nodewalk{#1,TeX={%
3618     \global\let\forest@global@temp\forest@nodewalk@historyback
3619     \global\let\forest@global@tempn\forest@nodewalk@n
3620   }}}%
3621   \forest@nodewalk@walklist{\forest@global@temp}{0}{\forest@global@tempn}{\let\forest@cn\forest@nodewalk@n}
3622 },
3623 define long step={walk and reverse}{n args=1,@bare,make for}{%
3624   \edef\forest@marshal{%
3625     \noexpand\pgfkeysalso{\unexpanded{#1}}%
3626     \noexpand\forest@nodewalk@walklist{\noexpand\forest@nodewalk@historyback}{0}{\noexpand\forest@nodewalk@n}
3627   }\forest@marshal
3628 },
3629 define long step={sort}{n args=1,@bare,make for}{%
3630   \forest@nodewalk{#1,TeX={%
3631     \global\let\forest@global@temp\forest@nodewalk@historyback
3632     \global\let\forest@global@tempn\forest@nodewalk@n
3633   }}}%
3634   \forest@nodewalk@sortlist{\forest@global@temp}{\forest@global@tempn}\forest@sort@ascending
3635 },
3636 define long step={sort'}{n args=1,@bare,make for}{%
3637   \forest@nodewalk{#1,TeX={%
3638     \global\let\forest@global@temp\forest@nodewalk@historyback
3639     \global\let\forest@global@tempn\forest@nodewalk@n
3640   }}}%
3641   \forest@nodewalk@sortlist{\forest@global@temp}{\forest@global@tempn}\forest@sort@descending
3642 },
3643 define long step={walk and sort}{n args=1,@bare,make for}{% walk as given, then walk sorted
3644   \edef\forest@marshal{%
3645     \noexpand\pgfkeysalso{\unexpanded{#1}}%
3646     \noexpand\forest@nodewalk@sortlist{\noexpand\forest@nodewalk@historyback}{\noexpand\forest@nodewalk@n}
3647   }\forest@marshal
3648 },
3649 define long step={walk and sort'}{n args=1,@bare,make for}{%
3650   \edef\forest@marshal{%
3651     \noexpand\pgfkeysalso{\unexpanded{#1}}%
3652     \noexpand\forest@nodewalk@sortlist{\noexpand\forest@nodewalk@historyback}{\noexpand\forest@nodewalk@n}
3653   }\forest@marshal

```

```

3654 },
3655 sort by/.store in=\forest@nodesort@by,
3656 define long step={save}{n args=2,@bare,make for}{% #1 = name, #2 = nodewalk
3657   \forest@forthis{%
3658     \forest@nodewalk{#2,TeX={%
3659       \global\let\forest@global@temp\forest@nodewalk@historyback
3660       \global\let\forest@global@tempn\forest@nodewalk@n
3661     }}{%
3662   }%
3663   \forest@nodewalk@walklist{ }\forest@global@temp{0}{\forest@global@tempn}\relax
3664   \csedef{forest@nodewalk@saved@#1}{\forest@nodewalk@walklist@walked}%
3665 },
3666 define long step={walk and save}{n args=2,@bare,make for}{% #1 = name, #2 = nodewalk
3667   \edef\forest@marshal{%
3668     \noexpand\pgfkeysalso{\unexpanded{#2}}%
3669     \noexpand\forest@nodewalk@walklist{ }\noexpand\forest@nodewalk@historyback{0}{\noexpand\forest@nodewalk
3670   }}\forest@marshal
3671   \csedef{forest@nodewalk@saved@#1}{\forest@nodewalk@walklist@walked}%
3672 },
3673 nodewalk/save history/.code 2 args={% #1 = back, forward
3674   \csedef{forest@nodewalk@saved@#1}{\forest@nodewalk@historyback}%
3675   \csedef{forest@nodewalk@saved@#2}{\forest@nodewalk@historyforward}%
3676 },
3677 define long step={load}{n args=1,@bare,make for}{%
3678   \forest@nodewalk@walklist{ }\csuse{forest@nodewalk@saved@#1}0,}{0}{-1}{\ifnum\forest@nodewalk@cn=0 \else\
3679 },
3680 if in saved nodewalk/.code n args=4{% is node #1 in nodewalk #2; yes: #3, no: #4
3681   \forest@forthis{%
3682     \forest@go{#1}%
3683     \edef\forest@marshal{%
3684       \noexpand\pgfutil@in@{,\forest@cn,}{,\csuse{forest@nodewalk@saved@#2},}%
3685     }\forest@marshal
3686   }%
3687   \ifpgfutil@in@
3688     \@escapeif{\pgfkeysalso{#3}}%
3689   \else
3690     \@escapeif{\pgfkeysalso{#4}}%
3691   \fi
3692 },
3693 where in saved nodewalk/.style n args=4{
3694   for tree={if in saved nodewalk={#1}{#2}{#3}{#4}}
3695 },
3696 nodewalk/options/.code={\forestset{#1}},
3697 nodewalk/TeX/.code={#1},
3698 nodewalk/TeX'/.code={\appto\forest@externalize@loadimages{#1}{#1},
3699 nodewalk/TeX''/.code={\appto\forest@externalize@loadimages{#1}},
3700 nodewalk/typeout/.style={TeX={\typeout{#1}}},
3701 % repeat is taken later from /forest/repeat
3702 }
3703 \def\forest@nodewalk@walklist#1#2#3#4#5{%
3704   % #1 = list of preceding, #2 = list to walk
3705   % #3 = from, #4 = to
3706   % #5 = every step code
3707   \let\forest@nodewalk@cn\forest@cn
3708   \edef\forest@marshal{%
3709     \noexpand\forest@nodewalk@walklist@{#1}{#2}{\number\numexpr#3}{\number\numexpr#4}{1}{0}{\unexpanded{#5}}%
3710   }\forest@marshal
3711 }
3712 \def\forest@nodewalk@walklist@#1#2#3#4#5#6#7{%
3713   % #1 = list of walked, #2 = list to walk
3714   % #3 = from, #4 = to

```

```

3715 % #5 = current step n, #6 = steps made
3716 % #7 = every step code
3717 \def\forest@nodewalk@walklist@walked{#1}%
3718 \def\forest@nodewalk@walklist@rest{#2}%
3719 \edef\forest@nodewalk@walklist@stepsmade{#6}%
3720 \ifnum#4<0
3721   \forest@temptrue
3722 \else
3723   \ifnum#5>#4\relax
3724     \forest@tempfalse
3725   \else
3726     \forest@temptrue
3727   \fi
3728 \fi
3729 \ifforest@temp
3730   \edef\forest@nodewalk@cn{\forest@csvlist@getfirst@{#2}}%
3731   \ifnum\forest@nodewalk@cn=0
3732     #7%
3733   \else
3734     \ifnum#5>#3\relax
3735       #7%
3736       \edef\forest@nodewalk@walklist@stepsmade{\number\numexpr#6+1}%
3737     \fi
3738     \forest@csvlist@getfirstrest@{#2}\forest@nodewalk@cn\forest@nodewalk@walklist@rest
3739     \@escapeifif{%
3740       \edef\forest@marshal{%
3741         \noexpand\forest@nodewalk@walklist@
3742         {\forest@nodewalk@cn,#1}{\forest@nodewalk@walklist@rest}{#3}{#4}{\number\numexpr#5+1}{\forest@nodewalk@walklist@stepsmade}%
3743       }\forest@marshal
3744     }%
3745   \fi
3746 \fi
3747 }
3748
3749 \def\forest@nodewalk@back@updatehistory{%
3750   \ifnum\forest@cn=0
3751     \let\forest@nodewalk@historyback\forest@nodewalk@walklist@rest
3752     \let\forest@nodewalk@historyforward\forest@nodewalk@walklist@walked
3753   \else
3754     \expandafter\forest@csvlist@getfirstrest@\expandafter{\forest@nodewalk@walklist@walked}\forest@temp\forest@nodewalk@back@updatehistory
3755     \edef\forest@nodewalk@historyback{\forest@temp,\forest@nodewalk@walklist@rest}%
3756   \fi
3757 }
3758 \def\forest@nodewalk@forward@updatehistory{%
3759   \let\forest@nodewalk@historyforward\forest@nodewalk@walklist@rest
3760   \let\forest@nodewalk@historyback\forest@nodewalk@walklist@walked
3761 }
3762 \def\forest@go#1{%
3763   \def\forest@nodewalk@config@everystep@method{independent}%
3764   \def\forest@nodewalk@config@history@method{inherited}%
3765   \def\forest@nodewalk@config@oninvalid{inherited}%
3766   \forest@saveandrestoremacro\forest@nodewalk@oninvalid{%
3767     \forest@Nodewalk{#1}{}%
3768   }%
3769 }
3770 \def\forest@csvlist@getfirst@#1{% assuming that the list is nonempty and finishes with a comma
3771   \forest@csvlist@getfirst@@#1\forest@csvlist@getfirst@@}
3772 \def\forest@csvlist@getfirst@@#1,#2\forest@csvlist@getfirst@@{#1}
3773 \def\forest@csvlist@getrest@#1{% assuming that the list is nonempty and finishes with a comma
3774   \forest@csvlist@getrest@@#1\forest@csvlist@getrest@@}
3775 \def\forest@csvlist@getrest@@#1,#2\forest@csvlist@getrest@@{#2}

```

```

3776 \def\forest@csvlist@getfirstrest@#1#2#3{% assuming that the list is nonempty and finishes with a comma
3777 % #1 = list, #2 = cs receiving first, #3 = cs receiving rest
3778 \forest@csvlist@getfirstrest@@#1\forest@csvlist@getfirstrest@@{#2}{#3}}
3779 \def\forest@csvlist@getfirstrest@@#1,#2\forest@csvlist@getfirstrest@@#3#4{%
3780 \def#3{#1}%
3781 \def#4{#2}%
3782 }
3783 \def\forest@nodewalk@stripfakesteps#1{%
3784 % go to the last valid node if the walk contained any nodes, otherwise restore the current node
3785 \edef\forest@marshal{%
3786 \unexpanded{#1}%
3787 \noexpand\ifnum\noexpand\forest@nodewalk@n=\forest@nodewalk@n\relax
3788 \def\noexpand\forest@cn{\forest@cn}%
3789 \noexpand\else
3790 \unexpanded{%
3791 \edef\forest@cn{%
3792 \expandafter\forest@csvlist@getfirst@\expandafter{\forest@nodewalk@historyback}}%
3793 }%
3794 }%
3795 \noexpand\fi
3796 }\forest@marshal
3797 }
3798 \def\forest@nodewalk@tolastvalid{%
3799 \ifnum\forest@cn=0
3800 \edef\forest@cn{\expandafter\forest@csvlist@getfirst@\expandafter{\forest@nodewalk@historyback}}%
3801 \ifnum\forest@cn=0
3802 \let\forest@cn\forest@nodewalk@origin
3803 \fi
3804 \fi
3805 }
3806 \def\forest@nodewalk@sortlist#1#2#3{%#1=list,#2=to,#3=asc/desc
3807 \edef\forest@nodewalksort@list{#1}%
3808 \expandafter\forest@nodewalk@sortlist@\expandafter{\number\numexpr#2}{#3}%
3809 }
3810 \def\forest@nodewalk@sortlist@#1#2{%#1=to,#2=asc/desc
3811 \safeloop
3812 \unless\ifnum\safeloopn>#1\relax
3813 \expandafter\forest@csvlist@getfirstrest@\expandafter{\forest@nodewalksort@list}\forest@nodewalksort@cn\fi
3814 \csedef{forest@nodesort@\safeloopn}{\forest@nodewalksort@cn}%
3815 \saferepeat
3816 \edef\forest@nodesort@sortkey{\forest@nodesort@by}%
3817 \forest@sort\forest@nodesort@cmpnodes\forest@nodesort@let#2{1}{#1}%
3818 \def\forest@nodewalksort@sorted{}%
3819 \safeloop
3820 \unless\ifnum\safeloopn>#1\relax
3821 \edef\forest@cn{\csname forest@nodesort@\safeloopn\endcsname}%
3822 \forest@nodewalk@makestep
3823 \saferepeat
3824 }

```

Find minimal/maximal node in a walk.

```

3825 \forestset{
3826 define long step={min}{n args=1,@bare,make for}{% the first min in the argument nodewalk
3827 \forest@nodewalk{#1,TeX={%
3828 \global\let\forest@global@temp\forest@nodewalk@historyback
3829 }}}%
3830 \forest@nodewalk@minmax{\forest@global@temp}{-1}{<}{\forest@nodewalk@minmax@node,}%
3831 },
3832 define long step={mins}{n args=1,@bare,make for}{% all mins in the argument nodewalk
3833 \forest@nodewalk{#1,TeX={%
3834 \global\let\forest@global@temp\forest@nodewalk@historyback

```

```

3835     }}{}%
3836     \forest@nodewalk@minmax{\forest@global@temp}{-1}{<}{\forest@nodewalk@minmax@nodes}%
3837 },
3838 define long step={walk and min}{n args=1,@bare}{%
3839     \edef\forest@marshal{%
3840         \noexpand\pgfkeysalso{\unexpanded{#1}}%
3841         \noexpand\forest@nodewalk@minmax{\noexpand\forest@nodewalk@historyback}{\noexpand\forest@nodewalk@n-\fo
3842     }\forest@marshal
3843 },
3844 define long step={walk and mins}{n args=1,@bare}{%
3845     \edef\forest@marshal{%
3846         \noexpand\pgfkeysalso{\unexpanded{#1}}%
3847         \noexpand\forest@nodewalk@minmax{\noexpand\forest@nodewalk@historyback}{\noexpand\forest@nodewalk@n-\fo
3848     }\forest@marshal
3849 },
3850 define long step={min in nodewalk}{@bare}{% find the first min in the preceding nodewalk, step to it
3851     \forest@nodewalk@minmax{\forest@nodewalk@historyback}{-1}{<}{\forest@nodewalk@minmax@node,}%
3852 },
3853 define long step={mins in nodewalk}{@bare}{% find mins in the preceding nodewalk, step to mins
3854     \forest@nodewalk@minmax{\forest@nodewalk@historyback}{-1}{<}{\forest@nodewalk@minmax@nodes}%
3855 },
3856 define long step={min in nodewalk'}{@bare}{% find the first min in the preceding nodewalk, step to min in h
3857     \forest@nodewalk@minmax{\forest@nodewalk@historyback}{-1}{<}{}%
3858 },
3859 %
3860 define long step={max}{n args=1,@bare,make for}{% the first max in the argument nodewalk
3861     \forest@nodewalk{#1,TeX={%
3862         \global\let\forest@global@temp\forest@nodewalk@historyback
3863     }}{}%
3864     \forest@nodewalk@minmax{\forest@global@temp}{-1}{>}{\forest@nodewalk@minmax@node,}%
3865 },
3866 define long step={maxs}{n args=1,@bare,make for}{% all maxs in the argument nodewalk
3867     \forest@nodewalk{#1,TeX={%
3868         \global\let\forest@global@temp\forest@nodewalk@historyback
3869     }}{}%
3870     \forest@nodewalk@minmax{\forest@global@temp}{-1}{>}{\forest@nodewalk@minmax@nodes}%
3871 },
3872 define long step={walk and max}{n args=1,@bare}{%
3873     \edef\forest@marshal{%
3874         \noexpand\pgfkeysalso{\unexpanded{#1}}%
3875         \noexpand\forest@nodewalk@minmax{\noexpand\forest@nodewalk@historyback}{\noexpand\forest@nodewalk@n-\fo
3876     }\forest@marshal
3877 },
3878 define long step={walk and maxs}{n args=1,@bare}{%
3879     \edef\forest@marshal{%
3880         \noexpand\pgfkeysalso{\unexpanded{#1}}%
3881         \noexpand\forest@nodewalk@minmax{\noexpand\forest@nodewalk@historyback}{\noexpand\forest@nodewalk@n-\fo
3882     }\forest@marshal
3883 },
3884 define long step={max in nodewalk}{@bare}{% find the first max in the preceding nodewalk, step to it
3885     \forest@nodewalk@minmax{\forest@nodewalk@historyback}{-1}{>}{\forest@nodewalk@minmax@node,}%
3886 },
3887 define long step={maxs in nodewalk}{@bare}{% find maxs in the preceding nodewalk, step to maxs
3888     \forest@nodewalk@minmax{\forest@nodewalk@historyback}{-1}{>}{\forest@nodewalk@minmax@nodes}%
3889 },
3890 define long step={max in nodewalk'}{@bare}{% find the first max in the preceding nodewalk, step to max in h
3891     \forest@nodewalk@minmax{\forest@nodewalk@historyback}{-1}{>}{}%
3892 },
3893 }
3894
3895 \def\forest@nodewalk@minmax#1#2#3#4{%

```

```

3896 % #1 = list of nodes
3897 % #2 = max index in list (start with 1)
3898 % #3 = min/max = ascending/descending = </>
3899 % #4 = how many results? 1 = {\forest@nodewalk@minmax@node,}, all={\forest@nodewalk@minmax@nodes}, walk in
3900 \edef\forest@nodesort@sortkey{\forest@nodesort@by}%
3901 \edef\forest@nodewalk@minmax@N{\number\numexpr#2}%
3902 \edef\forest@nodewalk@minmax@n{%
3903 \edef\forest@nodewalk@minmax@list{#1}%
3904 \def\forest@nodewalk@minmax@nodes}%
3905 \def\forest@nodewalk@minmax@node}%
3906 \ifdefempty{\forest@nodewalk@minmax@list}{%
3907 }{%
3908 \safeloop
3909 \expandafter\forest@csvglist@getfirstrest\expandafter{\forest@nodewalk@minmax@list}\forest@nodewalk@min
3910 \ifnum\forest@nodewalk@minmax@cn=0 \else
3911 \ifdefempty{\forest@nodewalk@minmax@node}{%
3912 \edef\forest@nodewalk@minmax@node{\forest@nodewalk@minmax@cn}%
3913 \edef\forest@nodewalk@minmax@nodes{\forest@nodewalk@minmax@cn,}%
3914 \edef\forest@nodewalk@minmax@n{\safeloopn}%
3915 }{%
3916 \csedef{\forest@nodesort@1}{\forest@nodewalk@minmax@node}%
3917 \csedef{\forest@nodesort@2}{\forest@nodewalk@minmax@cn}%
3918 \forest@nodesort@cmpnodes{2}{1}%
3919 \if=\forest@sort@cmp@result
3920 \edef\forest@nodewalk@minmax@node{\forest@nodewalk@minmax@cn}%
3921 \epreto\forest@nodewalk@minmax@nodes{\forest@nodewalk@minmax@cn,}%
3922 \edef\forest@nodewalk@minmax@n{\safeloopn}%
3923 \else
3924 \if#3\forest@sort@cmp@result
3925 \edef\forest@nodewalk@minmax@node{\forest@nodewalk@minmax@cn}%
3926 \edef\forest@nodewalk@minmax@nodes{\forest@nodewalk@minmax@cn,}%
3927 \edef\forest@nodewalk@minmax@n{\safeloopn}%
3928 \fi
3929 \fi
3930 }%
3931 \fi
3932 \ifdefempty{\forest@nodewalk@minmax@list}{\forest@tempfalse}{\forest@temptrue}%
3933 \ifnum\safeloopn=\forest@nodewalk@minmax@N\relax\forest@temptrue\fi
3934 \ifforest@temp
3935 \saferepeat
3936 \edef\forest@nodewalk@minmax@list{#4}%
3937 \ifdefempty\forest@nodewalk@minmax@list{%
3938 \forestset{nodewalk/jump back=\forest@nodewalk@minmax@n-1}% CHECK
3939 }{%
3940 \safeloop
3941 \expandafter\forest@csvglist@getfirstrest\expandafter{\forest@nodewalk@minmax@list}\forest@cn\forest@
3942 \forest@nodewalk@makestep
3943 \ifdefempty{\forest@nodewalk@minmax@list}{\forest@tempfalse}{\forest@temptrue}%
3944 \ifforest@temp
3945 \saferepeat
3946 }%
3947 }%
3948 }

```

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.

```

3949 \newtoks\forest@nodewalk@shortsteps@resolution
3950 \newif\ifforest@nodewalk@areshortsteps
3951 \pgfqkeys{/forest/nodewalk}{
3952 .unknown/.code={%

```

```

3953 \forest@nodewalk@aeshortstepsfalse
3954 \pgfkeysifdefined{/forest/\pgfkeyscurrentname/.cmd}{%
3955 }{%
3956 \ifx\pgfkeyscurrentvalue\pgfkeysnovalue@text % no value, so possibly short steps
3957 \forest@nodewalk@shortsteps@resolution{%
3958 \forest@nodewalk@aeshortstepstrue
3959 \expandafter\forest@nodewalk@shortsteps\pgfkeyscurrentname=====,% "=" and "," cannot be short st
3960 \fi
3961 }%
3962 \ifforest@nodewalk@aeshortsteps
3963 \@escapeif{\expandafter\pgfkeysalso\expandafter{\the\forest@nodewalk@shortsteps@resolution}}%
3964 \else
3965 \@escapeif{\pgfkeysalso{/forest/\pgfkeyscurrentname={#1}}}%
3966 \fi
3967 },
3968 }
3969 \def\forest@nodewalk@shortsteps{%
3970 \futurelet\forest@nodewalk@nexttoken\forest@nodewalk@shortsteps@
3971 }
3972 \def\forest@nodewalk@shortsteps@{%
3973 \ifx\forest@nodewalk@nexttoken=
3974 \let\forest@nodewalk@nextop\forest@nodewalk@shortsteps@end
3975 \else
3976 \ifx\forest@nodewalk@nexttoken\bgroup
3977 \letcs\forest@nodewalk@nextop{forest@shortstep@group}%
3978 \else
3979 \let\forest@nodewalk@nextop\forest@nodewalk@shortsteps@@
3980 \fi
3981 \fi
3982 \forest@nodewalk@nextop
3983 }
3984 \def\forest@nodewalk@shortsteps@@#1{%
3985 \ifcsdef{forest@shortstep@#1}{%
3986 \csname forest@shortstep@#1\endcsname
3987 }{%
3988 \forest@nodewalk@aeshortstepsfalse
3989 \forest@nodewalk@shortsteps@end
3990 }%
3991 }
3992 % in the following definitions:
3993 % #1 = short step
3994 % #2 = (long) step, or a style in /forest/nodewalk (taking n args)
3995 \csdef{forest@nodewalk@defshortstep@0@args}#1#2{%
3996 \csdef{forest@shortstep@#1}{%
3997 \apptotoks\forest@nodewalk@shortsteps@resolution{,#2}%
3998 \forest@nodewalk@shortsteps}}
3999 \csdef{forest@nodewalk@defshortstep@1@args}#1#2{%
4000 \csdef{forest@shortstep@#1}##1{%
4001 \edef\forest@marshal####1{#2}%
4002 \eapptotoks\forest@nodewalk@shortsteps@resolution{,\forest@marshal{##1}}%
4003 \forest@nodewalk@shortsteps}}
4004 \csdef{forest@nodewalk@defshortstep@2@args}#1#2{%
4005 \csdef{forest@shortstep@#1}##1##2{%
4006 \edef\forest@marshal####1####2{#2}%
4007 \apptotoks\forest@nodewalk@shortsteps@resolution{,\forest@marshal{##1}{##2}}%
4008 \forest@nodewalk@shortsteps}}
4009 \csdef{forest@nodewalk@defshortstep@3@args}#1#2{%
4010 \csdef{forest@shortstep@#1}##1##2##3{%
4011 \edef\forest@marshal####1####2####3{#2}%
4012 \eapptotoks\forest@nodewalk@shortsteps@resolution{,\forest@marshal{##1}{##2}{##3}}%
4013 \forest@nodewalk@shortsteps}}

```



```

4014 \csdef{forest@nodewalk@defshortstep@4@args}#1#2{%
4015   \csdef{forest@shortstep@#1}##1##2##3##4{%
4016     \edef\forest@marshal####1####2####3####4{#2}%
4017     \eapptotoks\forest@nodewalk@shortsteps@resolution{,\forest@marshal{##1}{##2}{##3}{##4}}%
4018     \forest@nodewalk@shortsteps}}
4019 \csdef{forest@nodewalk@defshortstep@5@args}#1#2{%
4020   \csdef{forest@shortstep@#1}##1##2##3##4##5{%
4021     \edef\forest@marshal####1####2####3####4####5{#2}%
4022     \eapptotoks\forest@nodewalk@shortsteps@resolution{,\forest@marshal{##1}{##2}{##3}{##4}{##5}}%
4023     \forest@nodewalk@shortsteps}}
4024 \csdef{forest@nodewalk@defshortstep@6@args}#1#2{%
4025   \csdef{forest@shortstep@#1}##1##2##3##4##5##6{%
4026     \edef\forest@marshal####1####2####3####4####5####6{#2}%
4027     \eapptotoks\forest@nodewalk@shortsteps@resolution{,\forest@marshal{##1}{##2}{##3}{##4}{##5}{##6}}%
4028     \forest@nodewalk@shortsteps}}
4029 \csdef{forest@nodewalk@defshortstep@7@args}#1#2{%
4030   \csdef{forest@shortstep@#1}##1##2##3##4##5##6##7{%
4031     \edef\forest@marshal####1####2####3####4####5####6####7{#2}%
4032     \eapptotoks\forest@nodewalk@shortsteps@resolution{,\forest@marshal{##1}{##2}{##3}{##4}{##5}{##6}{##7}}%
4033     \forest@nodewalk@shortsteps}}
4034 \csdef{forest@nodewalk@defshortstep@8@args}#1#2{%
4035   \csdef{forest@shortstep@#1}##1##2##3##4##5##6##7##8{%
4036     \edef\forest@marshal####1####2####3####4####5####6####7####8{#2}%
4037     \eapptotoks\forest@nodewalk@shortsteps@resolution{,\forest@marshal{##1}{##2}{##3}{##4}{##5}{##6}{##7}{##8}}%
4038     \forest@nodewalk@shortsteps}}
4039 \csdef{forest@nodewalk@defshortstep@9@args}#1#2{%
4040   \csdef{forest@shortstep@#1}##1##2##3##4##5##6##7##8##9{%
4041     \edef\forest@marshal####1####2####3####4####5####6####7####8####9{#2}%
4042     \eapptotoks\forest@nodewalk@shortsteps@resolution{,\forest@marshal{##1}{##2}{##3}{##4}{##5}{##6}{##7}{##8}{##9}}%
4043     \forest@nodewalk@shortsteps}}
4044 \forestset{
4045   define short step/.code n args=3{% #1 = short step, #2 = n args, #3 = long step
4046     \csname forest@nodewalk@defshortstep@#2@args\endcsname{##1}{##3}%
4047   },
4048 }
4049 \def\forest@nodewalk@shortsteps@end#1,{
  Define short-form steps.

4050 \forestset{
4051   define short step={group}{1}{group={#1}}, % {braces} are special
4052   define short step={p}{0}{previous},
4053   define short step={n}{0}{next},
4054   define short step={u}{0}{parent},
4055   define short step={s}{0}{sibling},
4056   define short step={c}{0}{current},
4057   define short step={o}{0}{origin},
4058   define short step={r}{0}{root},
4059   define short step={R}{0}{root'},
4060   define short step={P}{0}{previous leaf},
4061   define short step={N}{0}{next leaf},
4062   define short step={F}{0}{first leaf},
4063   define short step={L}{0}{last leaf},
4064   define short step={>}{0}{next on tier},
4065   define short step={<}{0}{previous on tier},
4066   define short step={1}{0}{n=1},
4067   define short step={2}{0}{n=2},
4068   define short step={3}{0}{n=3},
4069   define short step={4}{0}{n=4},
4070   define short step={5}{0}{n=5},
4071   define short step={6}{0}{n=6},
4072   define short step={7}{0}{n=7},

```

```

4073 define short step={8}{0}{n=8},
4074 define short step={9}{0}{n=9},
4075 define short step={1}{0}{last},
4076 define short step={b}{0}{back},
4077 define short step={f}{0}{forward},
4078 define short step={v}{0}{last valid},
4079 define short step={*}{2}{repeat={#1}{#2}},
4080 for 1/.style={for nodewalk={n=1}{#1}},
4081 for 2/.style={for nodewalk={n=2}{#1}},
4082 for 3/.style={for nodewalk={n=3}{#1}},
4083 for 4/.style={for nodewalk={n=4}{#1}},
4084 for 5/.style={for nodewalk={n=5}{#1}},
4085 for 6/.style={for nodewalk={n=6}{#1}},
4086 for 7/.style={for nodewalk={n=7}{#1}},
4087 for 8/.style={for nodewalk={n=8}{#1}},
4088 for 9/.style={for nodewalk={n=9}{#1}},
4089 for -1/.style={for nodewalk={n'=1}{#1}},
4090 for -2/.style={for nodewalk={n'=2}{#1}},
4091 for -3/.style={for nodewalk={n'=3}{#1}},
4092 for -4/.style={for nodewalk={n'=4}{#1}},
4093 for -5/.style={for nodewalk={n'=5}{#1}},
4094 for -6/.style={for nodewalk={n'=6}{#1}},
4095 for -7/.style={for nodewalk={n'=7}{#1}},
4096 for -8/.style={for nodewalk={n'=8}{#1}},
4097 for -9/.style={for nodewalk={n'=9}{#1}},
4098 }

```

Define multiple-step walks.

```

4099 \forestset{
4100   define long step={tree}{-}{\forest@node@foreach{\forest@nodewalk@makestep}},
4101   define long step={tree reversed}{-}{\forest@node@foreach@reversed{\forest@nodewalk@makestep}},
4102   define long step={tree children-first}{-}{\forest@node@foreach@childrenfirst{\forest@nodewalk@makestep}},
4103   define long step={tree children-first reversed}{-}{\forest@node@foreach@childrenfirst@reversed{\forest@nodewalk@makestep}},
4104   define long step={tree breadth-first}{-}{\forest@node@foreach@breadthfirst{-1}{\forest@nodewalk@makestep}},
4105   define long step={tree breadth-first reversed}{-}{\forest@node@foreach@breadthfirst@reversed{-1}{\forest@nodewalk@makestep}},
4106   define long step={descendants}{-}{\forest@node@foreachdescendant{\forest@nodewalk@makestep}},
4107   define long step={descendants reversed}{-}{\forest@node@foreachdescendant@reversed{\forest@nodewalk@makestep}},
4108   define long step={descendants children-first}{-}{\forest@node@foreachdescendant@childrenfirst{\forest@nodewalk@makestep}},
4109   define long step={descendants children-first reversed}{-}{\forest@node@foreachdescendant@childrenfirst@reversed{\forest@nodewalk@makestep}},
4110   define long step={descendants breadth-first}{-}{\forest@node@foreach@breadthfirst{0}{\forest@nodewalk@makestep}},
4111   define long step={descendants breadth-first reversed}{-}{\forest@node@foreach@breadthfirst@reversed{0}{\forest@nodewalk@makestep}},
4112   define long step={level}{n args=1}{%
4113     \pgfmathtruncatemacro\forest@temp{#1}%
4114     \edef\forest@marshal{%
4115       \noexpand\forest@node@foreach@breadthfirst
4116       {\forest@temp}%
4117       {\noexpand\ifnum\noexpand\forest@temp>{level}=\forest@temp\relax\noexpand\forest@nodewalk@makestep\noexpand\forest@temp-1}%
4118     }\forest@marshal
4119   },
4120   define long step={level>}{n args=1}{%
4121     \pgfmathtruncatemacro\forest@temp{#1}%
4122     \edef\forest@marshal{%
4123       \noexpand\forest@node@foreach@breadthfirst
4124       {-1}%
4125       {\noexpand\ifnum\noexpand\forest@temp>{level}<\forest@temp\relax\noexpand\else\noexpand\forest@nodewalk@makestep\noexpand\forest@temp-1}%
4126     }\forest@marshal
4127   },
4128   define long step={level<}{n args=1}{%
4129     \pgfmathtruncatemacro\forest@temp{(#1)-1}%
4130     \show\forest@temp
4131     \ifnum\forest@temp=-1

```

```

4132 % special case, as \forest@node@foreach@breadthfirst uses level<0 as a signal for unlimited max level
4133 \ifnum\forest@level=0
4134 \forest@nodewalk@makestep
4135 \fi
4136 \else
4137 \edef\forest@marshal{%
4138 \noexpand\forest@node@foreach@breadthfirst
4139 {\forest@temp}%
4140 {\noexpand\forest@nodewalk@makestep}%
4141 }\forest@marshal
4142 \fi
4143 },
4144 define long step={level reversed}{n args=1}{%
4145 \pgfmathtruncatemacro\forest@temp{#1}%
4146 \edef\forest@marshal{%
4147 \noexpand\forest@node@foreach@breadthfirst@reversed
4148 {\forest@temp}%
4149 {\noexpand\ifnum\noexpand\forest@level=\forest@temp\relax\noexpand\forest@nodewalk@makestep\noexpand
4150 }\forest@marshal
4151 },
4152 define long step={level reversed>}{n args=1}{%
4153 \pgfmathtruncatemacro\forest@temp{#1}%
4154 \edef\forest@marshal{%
4155 \noexpand\forest@node@foreach@breadthfirst@reversed
4156 {-1}%
4157 {\noexpand\ifnum\noexpand\forest@level<\forest@temp\relax\noexpand\else\noexpand\forest@nodewalk@
4158 }\forest@marshal
4159 },
4160 define long step={level reversed<}{n args=1}{%
4161 \pgfmathtruncatemacro\forest@temp{(#1)-1}%
4162 \edef\forest@marshal{%
4163 \noexpand\forest@node@foreach@breadthfirst@reversed
4164 {\forest@temp}%
4165 {\noexpand\forest@nodewalk@makestep}%
4166 }\forest@marshal
4167 },
4168 %
4169 define long step={relative level}{n args=1}{%
4170 \pgfmathtruncatemacro\forest@temp{(#1)+\forest@level}}%
4171 \edef\forest@marshal{%
4172 \noexpand\forest@node@foreach@breadthfirst
4173 {\forest@temp}%
4174 {\noexpand\ifnum\noexpand\forest@level=\forest@temp\relax\noexpand\forest@nodewalk@makestep\noexpand
4175 }\forest@marshal
4176 },
4177 define long step={relative level>}{n args=1}{%
4178 \pgfmathtruncatemacro\forest@temp{(#1)+\forest@level}}%
4179 \edef\forest@marshal{%
4180 \noexpand\forest@node@foreach@breadthfirst
4181 {-1}%
4182 {\noexpand\ifnum\noexpand\forest@level<\forest@temp\relax\noexpand\else\noexpand\forest@nodewalk@
4183 }\forest@marshal
4184 },
4185 define long step={relative level<}{n args=1}{%
4186 \pgfmathtruncatemacro\forest@temp{(#1)+\forest@level-1}%
4187 \edef\forest@marshal{%
4188 \noexpand\forest@node@foreach@breadthfirst
4189 {\forest@temp}%
4190 {\noexpand\forest@nodewalk@makestep}%
4191 }\forest@marshal
4192 },

```

```

4193 define long step={relative level reversed}{n args=1}{%
4194   \pgfmathtruncatemacro\forest@temp{(#1)+\forestove{level}}%
4195   \edef\forest@marshal{%
4196     \noexpand\forest@node@foreach@breadthfirst@reversed
4197     {\forest@temp}%
4198     {\noexpand\ifnum\noexpand\forestove{level}=\forest@temp\relax\noexpand\forest@nodewalk@makestep\noexp
4199   }\forest@marshal
4200 },
4201 define long step={relative level reversed>}{n args=1}{%
4202   \pgfmathtruncatemacro\forest@temp{(#1)+\forestove{level}}%
4203   \edef\forest@marshal{%
4204     \noexpand\forest@node@foreach@breadthfirst@reversed
4205     {-1}%
4206     {\noexpand\ifnum\noexpand\forestove{level}<\forest@temp\relax\noexpand\else\noexpand\forest@nodewalk@
4207   }\forest@marshal
4208 },
4209 define long step={relative level reversed<}{n args=1}{%
4210   \pgfmathtruncatemacro\forest@temp{(#1)+\forestove{level}-1}%
4211   \edef\forest@marshal{%
4212     \noexpand\forest@node@foreach@breadthfirst@reversed
4213     {\forest@temp}%
4214     {\noexpand\forest@nodewalk@makestep}%
4215   }\forest@marshal
4216 },
4217 define long step={children}{\forest@node@foreachchild{\forest@nodewalk@makestep}},
4218 define long step={children reversed}{\forest@node@foreachchild@reversed{\forest@nodewalk@makestep}},
4219 define long step={current and following siblings}{\forest@node@@forselfandfollowingsiblings{\forest@nodew
4220 define long step={following siblings}{style}{if nodewalk valid={next}{fake=next,current and following sibli
4221 define long step={current and preceding siblings}{\forest@node@@forselfandprecedingsiblings{\forest@nodew
4222 define long step={preceding siblings}{style}{if nodewalk valid={previous}{fake=previous,current and precedi
4223 define long step={current and following siblings reversed}{\forest@node@@forselfandfollowingsiblings@reve
4224 define long step={following siblings reversed}{style}{fake=next,current and following siblings reversed},
4225 define long step={current and preceding siblings reversed}{\forest@node@@forselfandprecedingsiblings@reve
4226 define long step={preceding siblings reversed}{style}{fake=previous,current and preceding siblings reversed
4227 define long step={siblings}{style}{for nodewalk'={preceding siblings},following siblings},
4228 define long step={siblings reversed}{style}{for nodewalk'={following siblings reversed},preceding siblings
4229 define long step={current and siblings}{style}{for nodewalk'={preceding siblings},current and following sib
4230 define long step={current and siblings reversed}{style}{for nodewalk'={current and following siblings rever
4231 define long step={ancestors}{style}{while={}{parent},last valid},
4232 define long step={current and ancestors}{style}{current,ancestors},
4233 define long step={following nodes}{style}{while={}{next node},last valid},
4234 define long step={preceding nodes}{style}{while={}{previous node},last valid},
4235 define long step={current and following nodes}{style}{current,following nodes},
4236 define long step={current and preceding nodes}{style}{current,preceding nodes},
4237 }
4238 \let\forest@nodewalkstephandler@styletrueorfalse\forest@nodewalkstephandler@styletrue

```

6.6 Dynamic tree

```

4239 \def\forest@last@node{0}
4240 \csdef{forest@nodewalk@samedynamic nodes}{}
4241 \def\forest@nodehandleby@name@nodewalk@or@bracket#1{%
4242   \ifx\pgfkeysnovalue#1%
4243     \edef\forest@last@node{\forest@node@Nametoid{forest@last@node}}%
4244   \else
4245     \forest@nodehandleby@nnb@checkfirst#1\forest@END
4246   \fi
4247 }
4248 \def\forest@nodehandleby@nnb@checkfirst#1#2\forest@END{%
4249   \ifx[#1%

```

```

4250 \forest@create@node{#1#2}%
4251 \cseappto{forest@nodewalk@saved@dynamic nodes}{\forest@last@node,}%
4252 \else
4253 \forest@forthis{%
4254 \forest@nameandgo{#1#2}%
4255 \ifnum\forest@cn=0
4256 \PackageError{forest}{Cannot use a dynamic key on the invalid node}{}%
4257 \fi
4258 \let\forest@last@node\forest@cn
4259 }%
4260 \fi
4261 }
4262 \def\forest@create@node#1{% #1=bracket representation
4263 \bracketParse{\forest@create@collectafterthought}%
4264 \forest@last@node=#1\forest@end@create@node
4265 }
4266 \def\forest@create@collectafterthought#1\forest@end@create@node{%
4267 \forest@node@Foreach{\forest@last@node}{%
4268 \forest@to{delay}{given options}%
4269 \forest@set{given options}{}%
4270 }%
4271 \forest@eappto{\forest@last@node}{delay}{,\unexpanded{#1}}%
4272 \forest@set{\forest@last@node}{given options}{delay={}}%
4273 }
4274 \def\forest@create@node@and@process@given@options#1{% #1=bracket representation
4275 \bracketParse{\forest@createandprocess@collectafterthought}%
4276 \forest@last@node=#1\forest@end@create@node
4277 }
4278 \def\forest@createandprocess@collectafterthought#1\forest@end@create@node{%
4279 \forest@node@Compute@numeric@ts@info{\forest@last@node}%
4280 \forest@saveandrestoremacro\forest@root{%
4281 \let\forest@root\forest@last@node
4282 \forest@set{process keylist=given options}%
4283 }%
4284 }
4285 \def\forest@saveandrestoremacro#1#2{% #1 = the (zero-arg) macro to save before and restore after processing c
4286 \edef\forest@marshal{%
4287 \unexpanded{#2}%
4288 \noexpand\def\noexpand#1{\expandonce{#1}}%
4289 }\forest@marshal
4290 }
4291 \def\forest@saveandrestoreifcs#1#2{% #1 = the if cs to save before and restore after processing code in #2
4292 \edef\forest@marshal{%
4293 \unexpanded{#2}%
4294 \ifbool{#1}{\noexpand\setbool{#1}{true}}{\noexpand\setbool{#1}{false}}
4295 }\forest@marshal
4296 }
4297 \def\forest@saveandrestoretoks#1#2{% #1 = the toks to save before and restore after processing code in #2
4298 \edef\forest@marshal{%
4299 \unexpanded{#2}%
4300 \noexpand#1{\the#1}%
4301 }\forest@marshal
4302 }
4303 \def\forest@saveandrestoreregister#1#2{% #1 = the register to save before and restore after processing code i
4304 \edef\forest@marshal{%
4305 \unexpanded{#2}%
4306 \noexpand\forest@rset{#1}{\forest@register{#1}}%
4307 }\forest@marshal
4308 }
4309 \forest@set{
4310 save and restore register/.code 2 args={%

```

```

4311 \forest@saveandrestorereregister{filter@ed}{%
4312 \pgfkeysalso{#2}%
4313 }%
4314 },
4315 }
4316 \def\forest@remove@node#1{%
4317 \ifforestdebugdynamics\forestdebug@dynamics{before removing #1}\fi
4318 \forest@node@Remove{#1}%
4319 }
4320 \def\forest@append@node#1#2{%
4321 \ifforestdebugdynamics\forestdebug@dynamics{before appending #2 to #1}\fi
4322 \forest@dynamic@circularitytest{#2}{#1}{append}%
4323 \forest@node@Remove{#2}%
4324 \forest@node@Append{#1}{#2}%
4325 }
4326 \def\forest@prepend@node#1#2{%
4327 \ifforestdebugdynamics\forestdebug@dynamics{before prepending #2 to #1}\fi
4328 \forest@dynamic@circularitytest{#2}{#1}{prepend}%
4329 \forest@node@Remove{#2}%
4330 \forest@node@Prepend{#1}{#2}%
4331 }
4332 \def\forest@insertafter@node#1#2{%
4333 \ifforestdebugdynamics\forestdebug@dynamics{before inserting #2 after #1}\fi
4334 \forest@node@Remove{#2}%
4335 \forest@node@Insertafter{\forest@ve{#1}{@parent}}{#2}{#1}%
4336 }
4337 \def\forest@insertbefore@node#1#2{%
4338 \ifforestdebugdynamics\forestdebug@dynamics{before inserting #2 before #1}\fi
4339 \forest@node@Remove{#2}%
4340 \forest@node@Insertbefore{\forest@ve{#1}{@parent}}{#2}{#1}%
4341 }
4342 \def\forest@set@root#1#2{%
4343 \ifforestdebugdynamics\forestdebug@dynamics{before setting #1 as root}\fi
4344 \def\forest@root{#2}%
4345 }
4346 \def\forest@dynamic@circularitytest#1#2#3{%
4347 % #1=potential ancestor, #2=potential descendant, #3=message prefix
4348 \ifnum#1=#2
4349 \forest@circularityerror{#1}{#2}{#3}%
4350 \else
4351 \forest@for@node{#1}{%
4352 \forest@if@ancestor@of{#2}{\forest@circularityerror{#1}{#2}{#3}}{%
4353 }%
4354 \fi
4355 }
4356 \def\forest@circularityerror#1#2#3{%
4357 \forestdebug@typeouttrees{\forest@temp}%
4358 \PackageError{forest}{#3ing node id=#1 to id=#2 would result in a circular tree\MessageBreak forest of ids:
4359 }%
4360 \def\forestdebug@dynamics#1{%
4361 \forestdebug@typeouttrees\forest@temp
4362 \typeout{#1: \forest@temp}%
4363 }
4364 \def\forest@appto@do@dynamics#1#2{%
4365 \forest@node@handleby@name@nodewalk@or@bracket{#2}%
4366 \ifcase\forest@dynamics@copyhow\relax\or
4367 \forest@tree@copy{\forest@last@node}\forest@last@node
4368 \or
4369 \forest@node@copy{\forest@last@node}\forest@last@node
4370 \fi
4371 \forest@node@if@named@defined{forest@last@node}{%

```

```

4372     \forestOpreto{\forest@last@node}{delay}
4373     {for id={\forest@node@Nametoid{\forest@last@node}}{alias=forest@last@node},}%
4374     }-}%
4375 \edef\forest@marshal{%
4376     \noexpand\apptotoks\noexpand\forest@do@ynamics{%
4377     \noexpand#1{\forest@cn}{\forest@last@node}}%
4378     }\forest@marshal
4379 }
4380 \forestset{%
4381     create/.code={%
4382     \forest@create@node{#1}%
4383     \forest@for@node{\forest@last@node}{%
4384     \forest@node@setalias{\forest@last@node}%
4385     \cseappto{\forest@nodewalk@saved@dynamic nodes}{\forest@last@node},}%
4386     }%
4387 },
4388     create'/.code={%
4389     \forest@create@node@and@process@given@options{#1}%
4390     \forest@for@node{\forest@last@node}{%
4391     \forest@node@setalias{\forest@last@node}%
4392     \cseappto{\forest@nodewalk@saved@dynamic nodes}{\forest@last@node},}%
4393     }%
4394 },
4395     append/.code={\def\forest@ynamics@copyhow{0}\forest@appto@do@ynamics\forest@append@node{#1}},
4396     prepend/.code={\def\forest@ynamics@copyhow{0}\forest@appto@do@ynamics\forest@prepend@node{#1}},
4397     insert after/.code={\def\forest@ynamics@copyhow{0}\forest@appto@do@ynamics\forest@insertafter@node{#1}},
4398     insert before/.code={\def\forest@ynamics@copyhow{0}\forest@appto@do@ynamics\forest@insertbefore@node{#1}},
4399     append'/.code={\def\forest@ynamics@copyhow{1}\forest@appto@do@ynamics\forest@append@node{#1}},
4400     prepend'/.code={\def\forest@ynamics@copyhow{1}\forest@appto@do@ynamics\forest@prepend@node{#1}},
4401     insert after'/.code={\def\forest@ynamics@copyhow{1}\forest@appto@do@ynamics\forest@insertafter@node{#1}},
4402     insert before'/.code={\def\forest@ynamics@copyhow{1}\forest@appto@do@ynamics\forest@insertbefore@node{#1}},
4403     append''/.code={\def\forest@ynamics@copyhow{2}\forest@appto@do@ynamics\forest@append@node{#1}},
4404     prepend''/.code={\def\forest@ynamics@copyhow{2}\forest@appto@do@ynamics\forest@prepend@node{#1}},
4405     insert after''/.code={\def\forest@ynamics@copyhow{2}\forest@appto@do@ynamics\forest@insertafter@node{#1}},
4406     insert before''/.code={\def\forest@ynamics@copyhow{2}\forest@appto@do@ynamics\forest@insertbefore@node{#1}},
4407     remove/.code={%
4408     \pgfkeysalso{alias=forest@last@node}%
4409     \cseappto{\forest@nodewalk@saved@dynamic nodes}{\forest@cn},}%
4410     \expandafter\apptotoks\expandafter\forest@do@ynamics\expandafter{%
4411     \expandafter\forest@remove@node\expandafter{\forest@cn}}%
4412 },
4413     set root/.code={%
4414     \def\forest@ynamics@copyhow{0}%
4415     \forest@appto@do@ynamics\forest@set@root{#1}%
4416 },
4417     replace by/.code={\forest@replaceby@code{#1}{insert after}},
4418     replace by'/.code={\forest@replaceby@code{#1}{insert after'}},
4419     replace by''/.code={\forest@replaceby@code{#1}{insert after''}},
4420     sort/.code={%
4421     \eapptotoks\forest@do@ynamics{%
4422     \noexpand\forest@nodesort
4423     \noexpand\forest@sort@ascending
4424     {\forest@cn}%
4425     {\expandonce{\forest@nodesort@by}}%
4426     }%
4427 },
4428     sort'/.code={%
4429     \eapptotoks\forest@do@ynamics{%
4430     \noexpand\forest@nodesort
4431     \noexpand\forest@sort@descending
4432     {\forest@cn}%

```

```

4433     {\expandonce{\forest@nodesort@by}}%
4434   }%
4435 },
4436 sort by/.store in=\forest@nodesort@by,
4437 }
4438 \def\forest@replaceby@code#1#2{%#1=node spec,#2=insert after['']['']}
4439 \ifnum\forest@tove{@parent}=0
4440   \cseappto{\forest@nodewalk@saved@dynamic nodes}{\forest@cn,}%
4441   \pgfkeysalso{alias=forest@last@node,set root={#1}}%
4442 \else
4443   \cseappto{\forest@nodewalk@saved@dynamic nodes}{\forest@cn,}%
4444   \pgfkeysalso{alias=forest@last@node,#2={#1}}%
4445   \eapptotoks\forest@do@ynamics{%
4446     \noexpand\ifnum\noexpand\forest@tve{\forest@cn}{@parent}=\forest@tve{@parent}
4447     \noexpand\forest@remove@node{\forest@cn}%
4448     \noexpand\fi
4449   }%
4450 \fi
4451 }
4452 \def\forest@nodesort#1#2#3{% #1 = direction, #2 = parent node, #3 = sort key
4453   \ifforestdebugdynamics\forestdebug@dynamics{before sorting children of #2}\fi
4454   \def\forest@nodesort@sortkey{#3}%
4455   \forest@for@node{#2}{\forest@nodesort@#1}%
4456   \ifforestdebugdynamics\forestdebug@dynamics{after sorting children of #2}\fi
4457 }
4458 \def\forest@nodesort@#1{%
4459   % prepare the array of child ids
4460   \c@pgf@counta=0
4461   \forest@toget{@first}\forest@nodesort@id
4462   \forest@loop
4463   \ifnum\forest@nodesort@id>0
4464     \advance\c@pgf@counta 1
4465     \csedef{\forest@nodesort@the\c@pgf@counta}{\forest@nodesort@id}%
4466     \forest@get{\forest@nodesort@id}{@next}\forest@nodesort@id
4467   \forest@repeat
4468   % sort
4469   \forest@toget{n children}\forest@nodesort@n
4470   \forest@sort\forest@nodesort@cmpnodes\forest@nodesort@let#1{1}{\forest@nodesort@n}%
4471   % remove all children
4472   \forest@toget{@first}\forest@nodesort@id
4473   \forest@loop
4474   \ifnum\forest@nodesort@id>0
4475     \forest@node@Remove{\forest@nodesort@id}%
4476     \forest@toget{@first}\forest@nodesort@id
4477   \forest@repeat
4478   % insert the children in new order
4479   \c@pgf@counta=0
4480   \forest@loop
4481   \ifnum\c@pgf@counta<\forest@nodesort@n\relax
4482     \advance\c@pgf@counta 1
4483     \edef\temp{\csname forest@nodesort@the\c@pgf@counta\endcsname}%
4484     \forest@node@append{\csname forest@nodesort@the\c@pgf@counta\endcsname}%
4485   \forest@repeat
4486 }
4487 \def\forest@nodesort@cmpnodes#1#2{%
4488   \global\let\forest@nodesort@cmpresult\forest@sort@cmp@eq
4489   \foreach \forest@temp@pgfmath in \forest@nodesort@sortkey {%
4490     \forest@for@node{\csname forest@nodesort@#1\endcsname}{%
4491       \pgfmathparse{\forest@temp@pgfmath}\global\let\forest@global@tempa\pgfmathresult}%
4492     \forest@for@node{\csname forest@nodesort@#2\endcsname}{%
4493       \pgfmathparse{\forest@temp@pgfmath}\global\let\forest@global@tempb\pgfmathresult}%

```



```

4494 \ifdim\forest@global@tempa pt<\forest@global@tempb pt
4495 \global\let\forest@nodesort@cmpresult\forest@sort@cmp@lt
4496 \breakforeach
4497 \else
4498 \ifdim\forest@global@tempa pt>\forest@global@tempb pt
4499 \global\let\forest@nodesort@cmpresult\forest@sort@cmp@gt
4500 \breakforeach
4501 \fi
4502 \fi
4503 }%
4504 \forest@nodesort@cmpresult
4505 }
4506 \def\forest@nodesort@let#1#2{%
4507 \csletcs{forest@nodesort@#1}{forest@nodesort@#2}%
4508 }
4509 \forestset{
4510 define long step={last dynamic node}{style,must start at valid node=false}{%
4511 name=forest@last@node
4512 }
4513 }

```

7 Stages

```

4514 \def\forest@root{0}
4515 %%% begin listing region: stages
4516 \forestset{
4517 stages/.style={
4518 for root'={
4519 process keylist register=default preamble,
4520 process keylist register=preamble
4521 },
4522 process keylist=given options,
4523 process keylist=before typesetting nodes,
4524 typeset nodes stage,
4525 process keylist=before packing,
4526 pack stage,
4527 process keylist=before computing xy,
4528 compute xy stage,
4529 process keylist=before drawing tree,
4530 draw tree stage
4531 },
4532 typeset nodes stage/.style={for root'=typeset nodes},
4533 pack stage/.style={for root'=pack},
4534 compute xy stage/.style={for root'=compute xy},
4535 draw tree stage/.style={for root'=draw tree},
4536 }
4537 %%% end listing region: stages
4538 \forestset{
4539 process keylist/.code={%
4540 \forest@process@hook@keylist{#1}{#1 processing order/.try,processing order/.lastretry}},
4541 process keylist'/ .code 2 args={\forest@process@hook@keylist@nodynamics{#1}{#2}},
4542 process keylist''/.code 2 args={\forest@process@hook@keylist@{#1}{#2}},
4543 process keylist register/.code={\forest@process@keylist@register{#1}},
4544 process delayed/.code={%
4545 \forest@havedelayedoptions{@delay}{#1}%
4546 \ifforest@havedelayedoptions
4547 \forest@process@hook@keylist@nodynamics{@delay}{#1}%
4548 \fi
4549 },
4550 do dynamics/.code={%
4551 \the\forest@do@dynamics

```

```

4552 \forest@do@ynamics{}%
4553 \forest@node@Compute@numeric@ts@info{\forest@root}%
4554 },
4555 declare keylist={given options}{},
4556 declare keylist={before typesetting nodes}{},
4557 declare keylist={before packing}{},
4558 declare keylist={before packing node}{},
4559 declare keylist={after packing node}{},
4560 declare keylist={before computing xy}{},
4561 declare keylist={before drawing tree}{},
4562 declare keylist={delay}{},
4563 delay n/.style 2 args={if={#1==0}{#2}{delay@n={#1}{#2}}},
4564 delay@n/.style 2 args={
4565   if={#1==1}{delay={#2}}{delay={delay@n/.process args={P}{#1-1}{#2}}}
4566 },
4567 if have delayed/.style 2 args={if have delayed'={processing order}{#1}{#2}},
4568 if have delayed'/.code n args=3{%
4569   \forest@havedelayedoptionsfalse
4570   \forest@forthis{%
4571     \forest@nodewalk{#1}{%
4572       TeX={%
4573         \forest@toget{delay}\forest@temp@delayed
4574         \ifdefempty\forest@temp@delayed{}{\forest@havedelayedoptionstrue}%
4575       }%
4576     }%
4577   }%
4578   \ifforest@havedelayedoptions\pgfkeysalso{#2}\else\pgfkeysalso{#3}\fi
4579 },
4580 typeset nodes/.code={%
4581   \forest@drawtree@preservenodeboxes@false
4582   \forest@nodewalk
4583     {typeset nodes processing order/.try,processing order/.lastretry}%
4584     {TeX={\forest@node@typeset}}%
4585 },
4586 typeset nodes'/.code={%
4587   \forest@drawtree@preservenodeboxes@true
4588   \forest@nodewalk
4589     {typeset nodes processing order/.try,processing order/.lastretry}%
4590     {TeX={\forest@node@typeset}}%
4591 },
4592 typeset node/.code={%
4593   \forest@drawtree@preservenodeboxes@false
4594   \forest@node@typeset
4595 },
4596 pack/.code={\forest@pack},
4597 pack'/.code={\forest@pack@onlythisnode},
4598 compute xy/.code={\forest@node@computeabsolute positions},
4599 draw tree box/.store in=\forest@drawtreebox,
4600 draw tree box,
4601 draw tree/.code={%
4602   \forest@drawtree@preservenodeboxes@false
4603   \forest@node@drawtree
4604 },
4605 draw tree'/.code={%
4606   \forest@drawtree@preservenodeboxes@true
4607   \forest@node@drawtree
4608 },
4609 %%% begin listing region: draw_tree_method
4610 draw tree method/.style={
4611   for nodewalk={
4612     draw tree nodes processing order/.try,

```

```

4613     draw tree processing order/.retry,
4614     processing order/.lastretry
4615   }{draw tree node},
4616   for nodewalk={
4617     draw tree edges processing order/.try,
4618     draw tree processing order/.retry,
4619     processing order/.lastretry
4620   }{draw tree edge},
4621   for nodewalk={
4622     draw tree tikz processing order/.try,
4623     draw tree processing order/.retry,
4624     processing order/.lastretry
4625   }{draw tree tikz}
4626 },
4627 %%% end listing region: draw_tree_method
4628 draw tree node/.code={\forest@draw@node},
4629 draw tree node'/.code={\forest@draw@node@},
4630 if node drawn/.code n args={3}{%
4631   \forest@forthis{%
4632     \forest@configured@nodewalk{independent}{inherited}{fake}{#1}{}%
4633     \ifnum\forest@cn=0
4634       \forest@tempfalse
4635     \else
4636       \ifcsdef{forest@drawn@\forest@cn}{\forest@temptrue}{\forest@tempfalse}%
4637     \fi
4638   }%
4639   \ifforest@temp\pgfkeysalso{#2}\else\pgfkeysalso{#3}\fi
4640 },
4641 draw tree edge/.code={\forest@draw@edge},
4642 draw tree edge'/.code={\forest@draw@edge@},
4643 draw tree tikz/.code={\forest@draw@tikz@}, % always!
4644 draw tree tikz'/.code={\forest@draw@tikz@},
4645 processing order/.nodewalk style={tree},
4646 %given options processing order/.style={processing order},
4647 %before typesetting nodes processing order/.style={processing order},
4648 %before packing processing order/.style={processing order},
4649 %before computing xy processing order/.style={processing order},
4650 %before drawing tree processing order/.style={processing order},
4651 }
4652 \newtoks\forest@do@dynamics
4653 \newif\ifforest@havedelayedoptions
4654 \def\forest@process@hook@keylist#1#2{%,#1=keylist,#2=processing order nodewalk
4655   \safeloop
4656     \forest@fornode{\forest@root}{\forest@process@hook@keylist@{#1}{#2}}%
4657     \expandafter\ifstrempy\expandafter{\the\forest@do@dynamics}{}%
4658     \the\forest@do@dynamics
4659     \forest@do@dynamics={}
4660     \forest@node@Compute@numeric@ts@info{\forest@root}%
4661   }%
4662   \forest@fornode{\forest@root}{\forest@havedelayedoptions{#1}{#2}}%
4663   \ifforest@havedelayedoptions
4664     \saferepeat
4665 }
4666 \def\forest@process@hook@keylist@nodynamics#1#2{%#1=keylist,#2=processing order nodewalk
4667   % note: this macro works on (nodewalk starting at) the current node
4668   \safeloop
4669     \forest@forthis{\forest@process@hook@keylist@{#1}{#2}}%
4670     \forest@havedelayedoptions{#1}{#2}%
4671     \ifforest@havedelayedoptions
4672       \saferepeat
4673 }

```

```

4674 \def\forest@process@hook@keylist@#1#2{%#1=keylist,#2=processing order nodewalk
4675   \forest@nodewalk{#2}{%
4676     TeX={%
4677       \forestoget{#1}\forest@temp@keys
4678       \ifdefvoid\forest@temp@keys{}{%
4679         \forestoset{#1}{}%
4680         \expandafter\forestset\expandafter{\forest@temp@keys}%
4681       }%
4682     }%
4683   }%
4684 }
4685 \def\forest@process@keylist@register#1{%
4686   \edef\forest@marshal{%
4687     \noexpand\forestset{\forestregister{#1}}%
4688   }\forest@marshal
4689 }

```

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

```

4690 \def\forest@havedelayedoptions#1#2{%#1 = keylist, #2=nodewalk
4691   \forest@havedelayedoptionsfalse
4692   \forest@forthis{%
4693     \forest@nodewalk{#2}{%
4694       TeX={%
4695         \forestoget{delay}\forest@temp@delayed
4696         \ifdefempty\forest@temp@delayed{}\forest@havedelayedoptionstrue}%
4697         \forestolet{#1}\forest@temp@delayed
4698         \forestoset{delay}{}%
4699       }%
4700     }%
4701   }%
4702 }

```

7.1 Typesetting nodes

```

4703 \def\forest@node@typeset{%
4704   \let\forest@next\forest@node@typeset@
4705   \forestoifdefined{@box}{%
4706     \forestoget{@box}\forest@temp
4707     \ifdefempty\forest@temp{%
4708       \locbox\forest@temp@box
4709       \forestolet{@box}\forest@temp@box
4710     }{%
4711       \ifforest@drawtree@preservenodeboxes@
4712       \let\forest@next\relax
4713       \fi
4714     }%
4715   }{%
4716     \locbox\forest@temp@box
4717     \forestolet{@box}\forest@temp@box
4718   }%
4719   \def\forest@node@typeset@restore{%
4720     \ifdefined\ifsa@tikz\forest@standalone@hack\fi
4721     \forest@next
4722     \forest@node@typeset@restore
4723   }
4724 \def\forest@standalone@hack{%
4725   \ifsa@tikz
4726     \let\forest@standalone@tikzpicture\tikzpicture
4727     \let\forest@standalone@endtikzpicture\endtikzpicture
4728     \let\tikzpicture\sa@orig@tikzpicture
4729     \let\endtikzpicture\sa@orig@endtikzpicture
4730   \def\forest@node@typeset@restore{%

```

```

4731 \let\tikzpicture\forest@standalone@tikzpicture
4732 \let\endtikzpicture\forest@standalone@endtikzpicture
4733 }%
4734 \fi
4735 }
4736 \newbox\forest@box
4737 \def\forest@pgf@notyetpositioned{not yet positionedPGFINTERNAL}
4738 \def\forest@node@typeset@{%
4739 \forestanchortotikzanchor{anchor}\forest@temp
4740 \edef\forest@marshal{%
4741 \noexpand\forestolet{anchor}\noexpand\forest@temp
4742 \noexpand\forest@node@typeset@@
4743 \noexpand\forestoset{anchor}{\forestov{anchor}}}%
4744 }\forest@marshal
4745 }
4746 \def\forest@node@typeset@@{%
4747 \forestoget{name}\forest@nodename
4748 \edef\forest@temp@nodeformat{\forestove{node format}}%
4749 \gdef\forest@smuggle{}%
4750 \setbox0=\hbox{%
4751 \begin{tikzpicture}%
4752 /forest/copy command key={/tikz/anchor}{/tikz/forest@orig@anchor},
4753 anchor/.style={%
4754 /forest/TeX={\forestanchortotikzanchor{##1}\forest@temp@anchor},
4755 forest@orig@anchor/.expand once=\forest@temp@anchor
4756 }]}
4757 \pgfpositionnodelater{\forest@positionnodelater@save}%
4758 \forest@temp@nodeformat
4759 \pgfinterruptpath
4760 \pgfpointanchor{\forest@pgf@notyetpositioned\forest@nodename}{\forestcomputenodeboundary}%
4761 \endpgfinterruptpath
4762 \end{tikzpicture}}%
4763 }%
4764 \setbox\forestove{@box}=\box\forest@box % smuggle the box
4765 \forestolet{@boundary}\forest@global@boundary
4766 \forest@smuggle % ... and the rest
4767 }
4768
4769
4770 \forestset{
4771 declare readonly dimen={min x}{0pt},
4772 declare readonly dimen={min y}{0pt},
4773 declare readonly dimen={max x}{0pt},
4774 declare readonly dimen={max y}{0pt},
4775 }
4776 \def\forest@patch@enormouscoordinateboxbounds@plus#1{%
4777 \expandafter\ifstrequal\expandafter{#1}{16000.0pt}{\edef#1{0.0\pgfmath@pt}}}%
4778 }
4779 \def\forest@patch@enormouscoordinateboxbounds@minus#1{%
4780 \expandafter\ifstrequal\expandafter{#1}{-16000.0pt}{\edef#1{0.0\pgfmath@pt}}}%
4781 }
4782 \def\forest@positionnodelater@save{%
4783 \global\setbox\forest@box=\box\pgfpositionnodelaterbox
4784 \xappto\forest@smuggle{\noexpand\forestoset{later@name}{\pgfpositionnodelatername}}%
4785 % a bug in pgf? ---well, here's a patch
4786 \forest@patch@enormouscoordinateboxbounds@plus\pgfpositionnodelaterminx
4787 \forest@patch@enormouscoordinateboxbounds@plus\pgfpositionnodelaterminy
4788 \forest@patch@enormouscoordinateboxbounds@minus\pgfpositionnodelatermaxx
4789 \forest@patch@enormouscoordinateboxbounds@minus\pgfpositionnodelatermaxy
4790 % end of patch
4791 \xappto\forest@smuggle{\noexpand\forestoset{min x}{\pgfpositionnodelaterminx}}%

```

```

4792 \xappto\forest@smuggle{\noexpand\forestset{min y}{\pgfpositionnodelaterminy}}%
4793 \xappto\forest@smuggle{\noexpand\forestset{max x}{\pgfpositionnodelatermaxx}}%
4794 \xappto\forest@smuggle{\noexpand\forestset{max y}{\pgfpositionnodelatermaxy}}%
4795 }
4796 \def\forest@node@forest@positionnodelater@restore{%
4797 \ifforest@drawtree@preservenodeboxes@
4798 \let\forest@boxorcopy\copy
4799 \else
4800 \let\forest@boxorcopy\box
4801 \fi
4802 \forestoget{@box}\forest@temp
4803 \setbox\pgfpositionnodelaterbox=\forest@boxorcopy\forest@temp
4804 \edef\pgfpositionnodelatername{\forestove{later@name}}%
4805 \edef\pgfpositionnodelaterminx{\forestove{min x}}%
4806 \edef\pgfpositionnodelaterminy{\forestove{min y}}%
4807 \edef\pgfpositionnodelatermaxx{\forestove{max x}}%
4808 \edef\pgfpositionnodelatermaxy{\forestove{max y}}%
4809 \ifforest@drawtree@preservenodeboxes@
4810 \else
4811 \forestset{@box}{}%
4812 \fi
4813 }

```

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.

```

4814 \def\forest@pack{%
4815 \pgfsyssoftpath@getcurrentpath\forest@pack@original@path
4816 \forest@pack@computetiers
4817 \forest@pack@computegrowthuniformity
4818 \forest@@@pack
4819 \pgfsyssoftpath@setcurrentpath\forest@pack@original@path
4820 }
4821 \def\forest@@@pack{%
4822 \ifnum\forestove{uniform growth}>0
4823 \ifnum\forestove{n children}>0
4824 \forest@pack@level@uniform
4825 \forest@pack@aligntiers@ofsubtree
4826 \forest@pack@sibling@uniform@recursive
4827 \fi
4828 \else
4829 \forest@node@foreachchild{\forest@@@pack}%
4830 \forest@process@hook@keylist@nodynamics{before packing node}{current}%
4831 \ifnum\forestove{n children}>0
4832 \forest@pack@level@nonuniform
4833 \forest@pack@aligntiers
4834 \forest@pack@sibling@uniform@applyreversed
4835 \fi
4836 \forestoget{after packing node}\forest@temp@keys
4837 \forest@process@hook@keylist@nodynamics{after packing node}{current}%
4838 \fi
4839 }
4840 % \forestset{recalculate tree boundary/.code={\forest@node@recalculate@edges}}
4841 \def\forest@node@recalculate@edges{%
4842 % \edef\forest@marshal{%
4843 % \noexpand\forest@forthis{\noexpand\forest@node@getedges{\forestove{grow}}}%
4844 % }\forest@marshal
4845 % }
4846 \def\forest@pack@onlythisnode{%
4847 \ifnum\forestove{n children}>0

```

```

4848 \forest@pack@computetiers
4849 \forest@pack@level@nonuniform
4850 \forest@pack@aligntiers
4851 \forest@node@foreachchild{\forestoset{s}{0\pgfmath@pt}}%
4852 \forest@pack@sibling@uniform@applyreversed
4853 \fi
4854 }

```

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

```

4855 \def\forest@pack@computegrowthuniformity{%
4856 \forest@node@foreachchild{\forest@pack@computegrowthuniformity}%
4857 \edef\forest@pack@cgu@uniformity{%
4858 \ifnum\forestove{n children}=0
4859 2\else 1\fi
4860 }%
4861 \foresttoget{grow}\forest@pack@cgu@parentgrow
4862 \forest@node@foreachchild{%
4863 \ifnum\forestove{uniform growth}=0
4864 \def\forest@pack@cgu@uniformity{0}%
4865 \else
4866 \ifnum\forestove{uniform growth}=1
4867 \ifnum\forestove{grow}=\forest@pack@cgu@parentgrow\relax\else
4868 \def\forest@pack@cgu@uniformity{0}%
4869 \fi
4870 \fi
4871 \fi
4872 }%
4873 \foresttoget{before packing node}\forest@temp@a
4874 \foresttoget{after packing node}\forest@temp@b
4875 \expandafter\expandafter\expandafter\ifstrempty\expandafter\expandafter\expandafter{\expandafter\forest@temp@a
4876 \forestolet{uniform growth}\forest@pack@cgu@uniformity
4877 }{%
4878 \forestoset{uniform growth}{0}%
4879 }%
4880 }

```

Pack children in the level dimension in a uniform tree.

```

4881 \def\forest@pack@level@uniform{%
4882 \let\forest@plu@minchildl\relax
4883 \foresttoget{grow}\forest@plu@grow
4884 \forest@node@foreachchild{%
4885 \forest@node@getboundingrectangle@ls{\forest@plu@grow}%
4886 \advance\pgf@xa\forestove{1}\relax
4887 \ifx\forest@plu@minchildl\relax
4888 \edef\forest@plu@minchildl{\the\pgf@xa}%
4889 \else
4890 \ifdim\pgf@xa<\forest@plu@minchildl\relax
4891 \edef\forest@plu@minchildl{\the\pgf@xa}%
4892 \fi
4893 \fi
4894 }%
4895 \forest@node@getboundingrectangle@ls{\forest@plu@grow}%
4896 \pgfutil@tempdima=\pgf@xb\relax
4897 \advance\pgfutil@tempdima -\forest@plu@minchildl\relax
4898 \advance\pgfutil@tempdima \forestove{1 sep}\relax
4899 \ifdim\pgfutil@tempdima>0pt
4900 \forest@node@foreachchild{%
4901 \forestoeset{1}{\the\dimexpr\forestove{1}+\the\pgfutil@tempdima}%
4902 }%
4903 \fi

```

```

4904 \forest@node@foreachchild{%
4905   \ifnum\forest@ve{n children}>0
4906     \forest@pack@level@uniform
4907   \fi
4908 }%
4909 }

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

4910 \def\forest@pack@level@nonuniform{%
4911   \let\forest@plu@minchildl\relax
4912   \forest@get{grow}\forest@plu@grow
4913   \forest@node@foreachchild{%
4914     \forest@node@getedge{negative}{\forest@plu@grow}{\forest@plnu@negativechildedge}%
4915     \forest@node@getedge{positive}{\forest@plu@grow}{\forest@plnu@positivechildedge}%
4916     \def\forest@plnu@childedge{\forest@plnu@negativechildedge\forest@plnu@positivechildedge}%
4917     \forest@path@getboundingrectangle@ls\forest@plnu@childedge{\forest@plu@grow}%
4918     \advance\pgf@xa\forest@ve{1}\relax
4919     \ifx\forest@plu@minchildl\relax
4920       \edef\forest@plu@minchildl{\the\pgf@xa}%
4921     \else
4922       \ifdim\pgf@xa<\forest@plu@minchildl\relax
4923         \edef\forest@plu@minchildl{\the\pgf@xa}%
4924       \fi
4925     \fi
4926   }%
4927   \forest@node@getboundingrectangle@ls{\forest@plu@grow}%
4928   \pgfutil@tempdima=\pgf@xb\relax
4929   \advance\pgfutil@tempdima -\forest@plu@minchildl\relax
4930   \advance\pgfutil@tempdima \forest@ve{1 sep}\relax
4931   \ifdim\pgfutil@tempdima>0pt
4932     \forest@node@foreachchild{%
4933       \forest@set{1}{\the\dimexpr\the\pgfutil@tempdima+\forest@ve{1}}%
4934     }%
4935   \fi
4936 }

Align tiers.

4937 \def\forest@pack@aligntiers{%
4938   \forest@get{grow}\forest@temp@parentgrow
4939   \forest@get{@tiers}\forest@temp@tiers
4940   \forlistloop\forest@pack@aligntier@\forest@temp@tiers
4941 }
4942 \def\forest@pack@aligntiers@ofsubtree{%
4943   \forest@node@foreach{\forest@pack@aligntiers}%
4944 }
4945 \def\forest@pack@aligntiers@computeabs1{%
4946   \forest@to{abs@1}{1}%
4947   \forest@node@foreachdescendant{\forest@pack@aligntiers@computeabs1}%
4948 }
4949 \def\forest@pack@aligntiers@computeabs1@{%
4950   \forest@set{abs@1}{\the\dimexpr\forest@ve{1}+\forest@ve{\forest@parent}}{abs@1}}%
4951 }
4952 \def\forest@pack@aligntier@#1{%
4953   \forest@pack@aligntiers@computeabs1
4954   \pgfutil@tempdima=-\maxdimen\relax
4955   \def\forest@temp@currenttier{#1}%
4956   \forest@node@foreach{%
4957     \forest@get{tier}\forest@temp@tier
4958     \ifx\forest@temp@currenttier\forest@temp@tier
4959       \ifdim\pgfutil@tempdima<\forest@ve{abs@1}\relax
4960         \pgfutil@tempdima=\forest@ve{abs@1}\relax
4961       \fi

```



```

4962   \fi
4963 }%
4964 \ifdim\pgfutil@tempdima=-\maxdimen\relax\else
4965   \forest@node@foreach{%
4966     \forest@get{tier}\forest@temp@tier
4967     \ifx\forest@temp@currenttier\forest@temp@tier
4968       \forest@set{1}{\the\dimexpr\pgfutil@tempdima-\forest@ve{abs@1}+\forest@ve{1}}%
4969     \fi
4970   }%
4971 \fi
4972 }

```

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

```

4973 \def\forest@pack@sibling@uniform@recursive{%
4974   \forest@node@foreachchild{\forest@pack@sibling@uniform@recursive}%
4975   \forest@pack@sibling@uniform@applyreversed
4976 }

```

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

```

4977 \def\forest@pack@sibling@uniform@applyreversed{%
4978   \ifnum\forest@ve{n children}>1
4979     \ifnum\forest@ve{reversed}=0
4980       \forest@pack@sibling@uniform@main{first}{last}{next}{previous}%
4981     \else
4982       \forest@pack@sibling@uniform@main{last}{first}{previous}{next}%
4983     \fi
4984   \else
4985     \ifnum\forest@ve{n children}=1

```

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

```

4986     \csname forest@calign@\forest@ve{calign}\endcsname
4987   \fi
4988 \fi
4989 }

```

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

```

4990 \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.

```

4991   \forest@get{@#1}\forest@child
4992   \edef\forest@marshal{%
4993     \noexpand\forest@for@node{\forest@ve{@#1}}{%
4994       \noexpand\forest@node@getedge
4995         {positive}%
4996       {\forest@ve{grow}}%
4997     \noexpand\forest@temp@edge
4998   }%
4999 } \forest@marshal
5000 \forest@pack@pgfpoint@child@position\forest@child
5001 \let\forest@previous@positive@edge\pgfutil@empty
5002 \forest@extendpath\forest@previous@positive@edge\forest@temp@edge{%
5003   \forest@get{\forest@child}{@#3}\forest@child

```

Loop until the current child is the null node.

```

5004   \edef\forest@previous@child@s{0\pgfmath@pt}%
5005   \safeloop
5006   \unless\ifnum\forest@child=0

```

Get the negative edge of the child.

```

5007 \edef\forest@temp{%
5008 \noexpand\forest@for@node{\forest@child}{%
5009 \noexpand\forest@node@getedge
5010 {negative}}%
5011 {\forest@ve{grow}}%
5012 \noexpand\forest@temp@edge
5013 }%
5014 }\forest@temp

```

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

```

5015 \forest@pack@pgfpoint@child@position\forest@child

```

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

```

5016 \let\forest@child@negative@edge\pgfutil@empty
5017 \forest@extendpath\forest@child@negative@edge\forest@temp@edge{}%

```

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

```

5018 \forest@setupgrowline{\forest@ve{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!)

```

5019 \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`.

```

5020 \ifx\forest@csdistance\relax
5021 %\forest@set{\forest@child}{s}{\forest@previous@child@s}%
5022 \else
5023 \advance\pgfutil@tempdimb-\forest@csdistance\relax
5024 \advance\pgfutil@tempdimb\forest@ve{s_sep}\relax
5025 \forest@set{\forest@child}{s}{\the\dimexpr\forest@ve{\forest@child}{s}-\forest@csdistance+\forest@ve{s}
5026 \fi

```

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

```

5027 \ifdim\forest@ve{\forest@child}{s}<\forest@previous@child@s\relax
5028 \forest@set{\forest@child}{s}{\forest@previous@child@s}%
5029 \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.

```

5030 \forest@get{\forest@child}{s}\forest@child@s
5031 \edef\forest@previous@child@s{\forest@child@s}%
5032 \edef\forest@temp{%
5033 \noexpand\forest@for@node{\forest@child}{%
5034 \noexpand\forest@node@getedge
5035 {positive}}%
5036 {\forest@ve{grow}}%
5037 \noexpand\forest@temp@edge
5038 }%
5039 }\forest@temp
5040 \forest@pack@pgfpoint@child@position\forest@child
5041 \forest@extendpath\forest@previous@positive@edge\forest@temp@edge{}%
5042 \forest@getpositivetightedgeofpath\forest@previous@positive@edge\forest@previous@positive@edge
5043 \forest@get{\forest@child}{#3}\forest@child
5044 \saferepeat

```

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

```

5045 \csname forest@calign@\forest@ve{calign}\endcsname
5046 }

```

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

```

5047 \def\forest@pack@pgfpoint@child@position#1{%
5048 {%

```

```

5049 \pgftransformreset
5050 \pgftransformrotate{\forestove{grow}}%
5051 \forest@for node{#1}{%
5052 \pgfpointransformed{\pgfqpoint{\forestove{1}}{\forestove{s}}}%
5053 }%
5054 }%
5055 }

Get the position of the node in the grow (#1)-rotated coordinate system.
5056 \def\forest@pack@pgfpoin@positioning grow#1{%
5057 {%
5058 \pgftransformreset
5059 \pgftransformrotate{#1}%
5060 \pgfpointransformed{\pgfqpoint{\forestove{1}}{\forestove{s}}}%
5061 }%
5062 }

Child alignment.
5063 \def\forest@calign@s@shift#1{%
5064 \pgfutil@tempdima=#1\relax
5065 \forest@node@foreach child{%
5066 \forestoeset{s}{\the\dimexpr\forestove{s}+\pgfutil@tempdima}%
5067 }%
5068 }
5069 \def\forest@calign@child{%
5070 \forest@calign@s@shift{-\forestOve{\forest@node@nornbarthchildid{\forestove{calign primary child}}}{s}}%
5071 }
5072 \csdef{forest@calign@child edge}{%
5073 {%
5074 \edef\forest@temp@child{\forest@node@nornbarthchildid{\forestove{calign primary child}}}%
5075 \pgftransformreset
5076 \pgftransformrotate{\forestove{grow}}%
5077 \pgfpointransformed{\pgfqpoint{\forestOve{\forest@temp@child}{1}}{\forestOve{\forest@temp@child}{s}}}%
5078 \pgf@xa=\pgf@x\relax\pgf@ya=\pgf@y\relax
5079 \forest@Pointanchor{\forest@temp@child}{child anchor}%
5080 \advance\pgf@xa\pgf@x\relax\advance\pgf@ya\pgf@y\relax
5081 \forest@pointanchor{parent anchor}%
5082 \advance\pgf@xa-\pgf@x\relax\advance\pgf@ya-\pgf@y\relax
5083 \edef\forest@marshal{%
5084 \noexpand\pgftransformreset
5085 \noexpand\pgftransformrotate{-\forestove{grow}}%
5086 \noexpand\pgfpointransformed{\noexpand\pgfqpoint{\the\pgf@xa}{\the\pgf@ya}}%
5087 }\forest@marshal
5088 }%
5089 \forest@calign@s@shift{\the\dimexpr-\the\pgf@y}%
5090 }
5091 \csdef{forest@calign@midpoint}{%
5092 \forest@calign@s@shift{\the\dimexpr Opt -%
5093 (\forestOve{\forest@node@nornbarthchildid{\forestove{calign primary child}}}{s}%
5094 +\forestOve{\forest@node@nornbarthchildid{\forestove{calign secondary child}}}{s}%
5095 )/2\relax
5096 }%
5097 }
5098 \csdef{forest@calign@edge midpoint}{%
5099 {%
5100 \edef\forest@temp@firstchild{\forest@node@nornbarthchildid{\forestove{calign primary child}}}%
5101 \edef\forest@temp@secondchild{\forest@node@nornbarthchildid{\forestove{calign secondary child}}}%
5102 \pgftransformreset
5103 \pgftransformrotate{\forestove{grow}}%
5104 \pgfpointransformed{\pgfqpoint{\forestOve{\forest@temp@firstchild}{1}}{\forestOve{\forest@temp@firstchild}{s}}}%
5105 \pgf@xa=\pgf@x\relax\pgf@ya=\pgf@y\relax
5106 \forest@Pointanchor{\forest@temp@firstchild}{child anchor}%

```

```

5107 \advance\pgf@xa\pgf@x\relax\advance\pgf@ya\pgf@y\relax
5108 \edef\forest@marshal{%
5109 \noexpand\pgfpointransformed{\noexpand\pgfqpoint{\forestOve{\forest@temp@secondchild}{1}}{\forestOve{\forest@temp@secondchild}{1}}}%
5110 }\forest@marshal
5111 \advance\pgf@xa\pgf@x\relax\advance\pgf@ya\pgf@y\relax
5112 \forest@Pointanchor{\forest@temp@secondchild}{child anchor}%
5113 \advance\pgf@xa\pgf@x\relax\advance\pgf@ya\pgf@y\relax
5114 \divide\pgf@xa2 \divide\pgf@ya2
5115 \forest@pointanchor{parent anchor}%
5116 \advance\pgf@xa-\pgf@x\relax\advance\pgf@ya-\pgf@y\relax
5117 \edef\forest@marshal{%
5118 \noexpand\pgftransformreset
5119 \noexpand\pgftransformrotate{-\forestove{grow}}}%
5120 \noexpand\pgfpointransformed{\noexpand\pgfqpoint{\the\pgf@xa}{\the\pgf@ya}}%
5121 }\forest@marshal
5122 }%
5123 \forest@calign@s@shift{\the\dimexpr-\the\pgf@y}%
5124 }

```

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.

```

5125 \csdef{forest@calign@fixed angles}{%
5126 \ifnum\forestove{n children}>1
5127 \edef\forest@ca@first@child{\forest@node@nornbarthchildid{\forestove{calign primary child}}}%
5128 \edef\forest@ca@second@child{\forest@node@nornbarthchildid{\forestove{calign secondary child}}}%
5129 \ifnum\forestove{reversed}=1
5130 \let\forest@temp\forest@ca@first@child
5131 \let\forest@ca@first@child\forest@ca@second@child
5132 \let\forest@ca@second@child\forest@temp
5133 \fi
5134 \forestOget{\forest@ca@first@child}{1}\forest@ca@first@l
5135 \forestOget{\forest@ca@second@child}{1}\forest@ca@second@l
5136 \pgfmathsetlengthmacro\forest@ca@desired@s@distance{%
5137 tan(\forestove{calign secondary angle})*\forest@ca@second@l
5138 -tan(\forestove{calign primary angle})*\forest@ca@first@l
5139 }%
5140 \forestOget{\forest@ca@first@child}{s}\forest@ca@first@s
5141 \forestOget{\forest@ca@second@child}{s}\forest@ca@second@s
5142 \pgfmathsetlengthmacro\forest@ca@actual@s@distance{%
5143 \forest@ca@second@s-\forest@ca@first@s}%
5144 \ifdim\forest@ca@desired@s@distance>\forest@ca@actual@s@distance\relax
5145 \ifdim\forest@ca@actual@s@distance=0pt
5146 \pgfmathsetlength\pgfutil@tempdima{tan(\forestove{calign primary angle})*\forest@ca@second@l}%
5147 \pgfmathsetlength\pgfutil@tempdimb{\forest@ca@desired@s@distance/(\forestove{n children}-1)}%
5148 \forest@node@foreachchild{%
5149 \forestoeset{s}{\the\pgfutil@tempdima}%
5150 \advance\pgfutil@tempdima\pgfutil@tempdimb
5151 }%
5152 \def\forest@calign@anchor{0pt}%
5153 \else
5154 \pgfmathsetmacro\forest@ca@ratio{%
5155 \forest@ca@desired@s@distance/\forest@ca@actual@s@distance}%
5156 \forest@node@foreachchild{%
5157 \pgfmathsetlengthmacro\forest@temp{\forest@ca@ratio*\forestove{s}}%
5158 \forestolet{s}\forest@temp
5159 }%
5160 \pgfmathsetlengthmacro\forest@calign@anchor{%
5161 -tan(\forestove{calign primary angle})*\forest@ca@first@l}%
5162 \fi

```

```

5163 \else
5164 \ifdim\forest@ca@desired@s@distance<\forest@ca@actual@s@distance\relax
5165 \pgfmithsetlengthmacro\forest@ca@ratio{%
5166 \forest@ca@actual@s@distance/\forest@ca@desired@s@distance}%
5167 \forest@node@foreachchild{%
5168 \pgfmithsetlengthmacro\forest@temp{\forest@ca@ratio*\forestove{1}}%
5169 \forestolet{1}\forest@temp
5170 }%
5171 \forestOget{\forest@ca@first@child}{1}\forest@ca@first@l
5172 \pgfmithsetlengthmacro\forest@calign@anchor{%
5173 -tan(\forestove{calign primary angle})*\forest@ca@first@l}%
5174 \fi
5175 \fi
5176 \forest@calign@s@shift{-\forest@calign@anchor}%
5177 \fi
5178 }
5179 \csdef{forest@calign@fixed edge angles}{%
5180 \ifnum\forestove{n children}>1
5181 \edef\forest@ca@first@child{\forest@node@nornbarthchildid{\forestove{calign primary child}}}%
5182 \edef\forest@ca@second@child{\forest@node@nornbarthchildid{\forestove{calign secondary child}}}%
5183 \ifnum\forestove{reversed}=1
5184 \let\forest@temp\forest@ca@first@child
5185 \let\forest@ca@first@child\forest@ca@second@child
5186 \let\forest@ca@second@child\forest@temp
5187 \fi
5188 \forestOget{\forest@ca@first@child}{1}\forest@ca@first@l
5189 \forestOget{\forest@ca@second@child}{1}\forest@ca@second@l
5190 \forest@pointanchor{parent anchor}%
5191 \edef\forest@ca@parent@anchor@s{\the\pgf@x}%
5192 \edef\forest@ca@parent@anchor@l{\the\pgf@y}%
5193 \forest@Pointanchor{\forest@ca@first@child}{child anchor}%
5194 \edef\forest@ca@first@child@anchor@s{\the\pgf@x}%
5195 \edef\forest@ca@first@child@anchor@l{\the\pgf@y}%
5196 \forest@Pointanchor{\forest@ca@second@child}{child anchor}%
5197 \edef\forest@ca@second@child@anchor@s{\the\pgf@x}%
5198 \edef\forest@ca@second@child@anchor@l{\the\pgf@y}%
5199 \pgfmithsetlengthmacro\forest@ca@desired@second@edge@s{tan(\forestove{calign secondary angle})*%
5200 (\forest@ca@second@l-\forest@ca@second@child@anchor@l+\forest@ca@parent@anchor@l)}%
5201 \pgfmithsetlengthmacro\forest@ca@desired@first@edge@s{tan(\forestove{calign primary angle})*%
5202 (\forest@ca@first@l-\forest@ca@first@child@anchor@l+\forest@ca@parent@anchor@l)}%
5203 \pgfmithsetlengthmacro\forest@ca@desired@s@distance{\forest@ca@desired@second@edge@s-\forest@ca@desired@first@edge@s}%
5204 \forestOget{\forest@ca@first@child}{s}\forest@ca@first@s
5205 \forestOget{\forest@ca@second@child}{s}\forest@ca@second@s
5206 \pgfmithsetlengthmacro\forest@ca@actual@s@distance{%
5207 \forest@ca@second@s+\forest@ca@second@child@anchor@s
5208 -\forest@ca@first@s-\forest@ca@first@child@anchor@s}%
5209 \ifdim\forest@ca@desired@s@distance>\forest@ca@actual@s@distance\relax
5210 \ifdim\forest@ca@actual@s@distance=0pt
5211 \foresttoget{n children}\forest@temp@n@children
5212 \forest@node@foreachchild{%
5213 \forest@pointanchor{child anchor}%
5214 \edef\forest@temp@child@anchor@s{\the\pgf@x}%
5215 \pgfmithsetlengthmacro\forest@temp{%
5216 \forest@ca@desired@first@edge@s+(\forestove{n}-1)*\forest@ca@desired@s@distance/(\forest@temp@n@children)}%
5217 \forestolet{s}\forest@temp
5218 }%
5219 \def\forest@calign@anchor{0pt}%
5220 \else
5221 \pgfmithsetmacro\forest@ca@ratio{%
5222 \forest@ca@desired@s@distance/\forest@ca@actual@s@distance}%
5223 \forest@node@foreachchild{%

```

```

5224     \forest@pointanchor{child anchor}%
5225     \edef\forest@temp@child@anchor@s{\the\pgf@x}%
5226     \pgfmathsetlengthmacro\forest@temp{%
5227         \forest@ca@ratio*(%
5228             \forestove{s}-\forest@ca@first@s
5229             +\forest@temp@child@anchor@s-\forest@ca@first@child@anchor@s)%
5230             +\forest@ca@first@s
5231             +\forest@ca@first@child@anchor@s-\forest@temp@child@anchor@s}%
5232     \forestolet{s}\forest@temp
5233 }%
5234 \pgfmathsetlengthmacro\forest@calign@anchor{%
5235     -tan(\forestove{calign primary angle})*(\forest@ca@first@l-\forest@ca@first@child@anchor@l+\forest@
5236     +\forest@ca@first@child@anchor@s-\forest@ca@parent@anchor@s
5237 }%
5238 \fi
5239 \else
5240 \ifdim\forest@ca@desired@s@distance<\forest@ca@actual@s@distance\relax
5241 \pgfmathsetlengthmacro\forest@ca@ratio{%
5242     \forest@ca@actual@s@distance/\forest@ca@desired@s@distance}%
5243 \forest@node@foreachchild{%
5244     \forest@pointanchor{child anchor}%
5245     \edef\forest@temp@child@anchor@l{\the\pgf@y}%
5246     \pgfmathsetlengthmacro\forest@temp{%
5247         \forest@ca@ratio*(%
5248             \forestove{l}+\forest@ca@parent@anchor@l-\forest@temp@child@anchor@l)
5249             -\forest@ca@parent@anchor@l+\forest@temp@child@anchor@l}%
5250     \forestolet{l}\forest@temp
5251 }%
5252 \forest@get{\forest@ca@first@child}{l}\forest@ca@first@l
5253 \pgfmathsetlengthmacro\forest@calign@anchor{%
5254     -tan(\forestove{calign primary angle})*(\forest@ca@first@l+\forest@ca@parent@anchor@l-\forest@temp@
5255     +\forest@ca@first@child@anchor@s-\forest@ca@parent@anchor@s
5256 }%
5257 \fi
5258 \fi
5259 \forest@calign@s@shift{-\forest@calign@anchor}%
5260 \fi
5261 }

```

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.

```

5262 \def\forest@node@getedge#1#2#3{%
5263     \forest@get{#1@edge@#2}#3%
5264     \ifx#3\relax
5265         \forest@node@foreachchild{%
5266             \forest@node@getedge{#1}{#2}{\forest@temp@edge}%
5267         }%
5268         \forest@forthis{\forest@node@getedges{#2}}%
5269         \forest@get{#1@edge@#2}#3%
5270     \fi
5271 }

```

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.

```

5272 \def\forest@node@getedges#1{%

```

Run the computation in a \TeX group.

```

5273     %{}

```

Setup the grow line.

```

5274     \forest@setupgrowline{#1}%

```

Get the edge of the node itself.

```

5275 \ifnum\forestove{ignore}=0
5276 \foresttoget{@boundary}\forest@node@boundary
5277 \else
5278 \def\forest@node@boundary{}%
5279 \fi
5280 \csname forest@getboth\forestove{fit}edgesofpath\endcsname
5281 \forest@node@boundary\forest@negative@node@edge\forest@positive@node@edge
5282 \forestolet{negative@edge@#1}\forest@negative@node@edge
5283 \forestolet{positive@edge@#1}\forest@positive@node@edge

```

Add the edges of the children.

```

5284 \get@edges@merge{negative}{#1}%
5285 \get@edges@merge{positive}{#1}%
5286 }%
5287 }

```

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

```

5288 \def\get@edges@merge#1#2{%
5289 \ifnum\forestove{n children}>0
5290 \foresttoget{#1@edge@#2}\forest@node@edge

```

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

```

5291 \forest@pointanchor{parent anchor}%
5292 \edef\forest@getedge@pa@1{\the\pgf@x}%
5293 \edef\forest@getedge@pa@s{\the\pgf@y}%
5294 \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).

```

5295 \pgfgettransform\forest@temp@transform
5296 \pgftransformreset
5297 \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.

```

5298 \def\forest@all@edges{}%
5299 \forest@node@foreachchild{%
5300 \foresttoget{#1@edge@#2}\forest@temp@edge
5301 \pgfpointransformed{\pgfpoint{\forestove{l}}{\forestove{s}}}%
5302 \forest@extendpath\forest@node@edge\forest@temp@edge}%
5303 \ifnum\forestove{ignore edge}=0
5304 \pgfpointadd
5305 {\pgfpointransformed{\pgfpoint{\forestove{l}}{\forestove{s}}}}%
5306 {\forest@pointanchor{child anchor}}%
5307 \pgfgetlastxy{\forest@getedge@ca@1}{\forest@getedge@ca@s}%
5308 \eappto\forest@all@edges{%
5309 \noexpand\pgfsyssoftpath@movetotoken{\forest@getedge@pa@1}{\forest@getedge@pa@s}%
5310 \noexpand\pgfsyssoftpath@linetotoken{\forest@getedge@ca@1}{\forest@getedge@ca@s}%
5311 }%
5312 % this deals with potential overlap of the edges:
5313 \eappto\forest@node@edge{\noexpand\pgfsyssoftpath@movetotoken{\forest@getedge@ca@1}{\forest@getedge@ca@s}}
5314 \fi
5315 }%
5316 \ifdefempty{\forest@all@edges}{}%
5317 \pgfintersectionofpaths{\pgfsetpath\forest@all@edges}{\pgfsetpath\forest@node@edge}%
5318 \def\forest@edgenode@intersections{}%
5319 \forest@merge@intersectionloop
5320 \eappto\forest@node@edge{\expandonce{\forest@all@edges}\expandonce{\forest@edgenode@intersections}}%
5321 }%
5322 \pgfsettransform\forest@temp@transform

```

Process the path into an edge and store the edge.

```

5323 \csname forest@get#1\forestove{fit}edgeofpath\endcsname\forest@node@edge\forest@node@edge
5324 \forestolet{#1@edge@#2}\forest@node@edge
5325 \fi
5326 }
5327 %\newloop\forest@merge@loop
5328 \def\forest@merge@intersectionloop{%
5329 \c@pgf@counta=0
5330 \forest@loop
5331 \ifnum\c@pgf@counta<\pgfintersectionsolutions\relax
5332 \advance\c@pgf@counta1
5333 \pgfpointintersectionsolution{\the\c@pgf@counta}%
5334 \eappto\forest@edgenode@intersections{\noexpand\pgfsyssoftpath@movetotoken
5335 {\the\pgf@x}{\the\pgf@y}}%
5336 \forest@repeat
5337 }

```

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

```

5338 \def\forest@node@getboundingrectangle@ls#1{%
5339 \forestoget{@boundary}\forest@node@boundary
5340 \forest@path@getboundingrectangle@ls\forest@node@boundary{#1}%
5341 }

```

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).

```

5342 \def\forest@pgfpathtransformed#1{%
5343 \forest@save@pgfsyssoftpath@tokendefs
5344 \let\pgfsyssoftpath@movetotoken\forest@pgfpathtransformed@moveto
5345 \let\pgfsyssoftpath@linetotoken\forest@pgfpathtransformed@lineto
5346 \pgfsyssoftpath@setcurrentpath\pgfutil@empty
5347 #1%
5348 \forest@restore@pgfsyssoftpath@tokendefs
5349 }
5350 \def\forest@pgfpathtransformed@moveto#1#2{%
5351 \forest@pgfpathtransformed@op\pgfsyssoftpath@moveto{#1}{#2}%
5352 }
5353 \def\forest@pgfpathtransformed@lineto#1#2{%
5354 \forest@pgfpathtransformed@op\pgfsyssoftpath@lineto{#1}{#2}%
5355 }
5356 \def\forest@pgfpathtransformed@op#1#2#3{%
5357 \pgfpointransformed{\pgfqpoint{#2}{#3}}%
5358 \edef\forest@temp{%
5359 \noexpand#1{\the\pgf@x}{\the\pgf@y}%
5360 }%
5361 \forest@temp
5362 }

```

7.2.1 Tiers

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

```

5363 \def\forest@pack@computetiers{%
5364 {%
5365 \forest@pack@tiers@getalltiersinsubtree
5366 \forest@pack@tiers@computetierhierarchy
5367 \forest@pack@tiers@findcontainers
5368 \forest@pack@tiers@raisecontainers
5369 \forest@pack@tiers@computeprocessingorder
5370 \gdef\forest@smuggle{}%
5371 \forest@pack@tiers@write
5372 }%
5373 \forest@node@foreach{\forestoset{@tiers}{}}%

```



```

5374 \forest@smuggle
5375 }

```

Puts all tiers contained in the subtree into attribute tiers.

```

5376 \def\forest@pack@tiers@getalltiersinsubtree{%
5377   \ifnum\forest@ve{n children}>0
5378     \forest@node@foreachchild{\forest@pack@tiers@getalltiersinsubtree}%
5379   \fi
5380   \forest@toget{tier}\forest@temp@mytier
5381   \def\forest@temp@mytiers{}%
5382   \ifdefempty\forest@temp@mytier{}{%
5383     \list@add\forest@temp@mytiers\forest@temp@mytier
5384   }%
5385   \ifnum\forest@ve{n children}>0
5386     \forest@node@foreachchild{%
5387       \forest@toget{tiers}\forest@temp@tiers
5388       \forlistloop\forest@pack@tiers@forhandlerA\forest@temp@tiers
5389     }%
5390   \fi
5391   \forest@tolet{tiers}\forest@temp@mytiers
5392 }
5393 \def\forest@pack@tiers@forhandlerA#1{%
5394   \ifinlist{#1}\forest@temp@mytiers{}{%
5395     \list@add\forest@temp@mytiers{#1}%
5396   }%
5397 }

```

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.

```

5398 \def\forest@pack@tiers@computetierhierarchy{%
5399   \def\forest@tiers@ancestors{}%
5400   \forest@toget{tiers}\forest@temp@mytiers
5401   \forlistloop\forest@pack@tiers@cth@init\forest@temp@mytiers
5402   \forest@pack@tiers@computetierhierarchy@
5403 }
5404 \def\forest@pack@tiers@cth@init#1{%
5405   \csdef{forest@tiers@higher@#1}{}%
5406   \csdef{forest@tiers@lower@#1}{}%
5407 }
5408 \def\forest@pack@tiers@computetierhierarchy@{%
5409   \forest@toget{tier}\forest@temp@mytier
5410   \ifdefempty\forest@temp@mytier{}{%
5411     \forlistloop\forest@pack@tiers@forhandlerB\forest@tiers@ancestors
5412     \list@add\forest@tiers@ancestors\forest@temp@mytier
5413   }%
5414   \forest@node@foreachchild{%
5415     \forest@pack@tiers@computetierhierarchy@
5416   }%
5417   \forest@toget{tier}\forest@temp@mytier
5418   \ifdefempty\forest@temp@mytier{}{%
5419     \forest@list@del\forest@tiers@ancestors\forest@temp@mytier
5420   }%
5421 }
5422 \def\forest@pack@tiers@forhandlerB#1{%
5423   \def\forest@temp@tier{#1}%
5424   \ifx\forest@temp@tier\forest@temp@mytier
5425     \PackageError{forest}{Circular tier hierarchy (tier \forest@temp@mytier)}{ }%
5426   \fi
5427   \ifinlistcs{#1}{forest@tiers@higher@\forest@temp@mytier}{ }{%
5428     \listcsadd{forest@tiers@higher@\forest@temp@mytier}{#1}%
5429   \xifinlistcs\forest@temp@mytier{forest@tiers@lower@#1}{ }{%

```

```

5430 \listcseadd{forest@tiers@lower@#1}{\forest@temp@mytier}}%
5431 }
5432 \def\forest@pack@tiers@findcontainers{%
5433 \forestoget{tiers}\forest@temp@tiers
5434 \forlistloop\forest@pack@tiers@findcontainer\forest@temp@tiers
5435 }
5436 \def\forest@pack@tiers@findcontainer#1{%
5437 \def\forest@temp@tier{#1}%
5438 \forestoget{tier}\forest@temp@mytier
5439 \ifx\forest@temp@tier\forest@temp@mytier
5440 \csedef{forest@tiers@container@#1}{\forest@cn}%
5441 \else\@escapeif{%
5442 \forest@pack@tiers@findcontainerA{#1}%
5443 }\fi%
5444 }
5445 \def\forest@pack@tiers@findcontainerA#1{%
5446 \c@pgf@counta=0
5447 \forest@node@foreachchild{%
5448 \forestoget{tiers}\forest@temp@tiers
5449 \ifinlist{#1}\forest@temp@tiers{%
5450 \advance\c@pgf@counta 1
5451 \let\forest@temp@child\forest@cn
5452 }}%
5453 }%
5454 \ifnum\c@pgf@counta>1
5455 \csedef{forest@tiers@container@#1}{\forest@cn}%
5456 \else\@escapeif{% surely =1
5457 \forest@fornode{\forest@temp@child}{%
5458 \forest@pack@tiers@findcontainer{#1}%
5459 }%
5460 }\fi
5461 }
5462 \def\forest@pack@tiers@raisecontainers{%
5463 \forestoget{tiers}\forest@temp@mytiers
5464 \forlistloop\forest@pack@tiers@rc@forhandlerA\forest@temp@mytiers
5465 }
5466 \def\forest@pack@tiers@rc@forhandlerA#1{%
5467 \edef\forest@tiers@temptier{#1}%
5468 \letcs\forest@tiers@containernodeoftier{forest@tiers@container@#1}%
5469 \letcs\forest@temp@lowertiers{forest@tiers@lower@#1}%
5470 \forlistloop\forest@pack@tiers@rc@forhandlerB\forest@temp@lowertiers
5471 }
5472 \def\forest@pack@tiers@rc@forhandlerB#1{%
5473 \letcs\forest@tiers@containernodeoflowertier{forest@tiers@container@#1}%
5474 \forestOget{\forest@tiers@containernodeoflowertier}{content}\lowercontent
5475 \forestOget{\forest@tiers@containernodeoftier}{content}\uppercontent
5476 \forest@fornode{\forest@tiers@containernodeoflowertier}{%
5477 \forest@ifancestorof
5478 {\forest@tiers@containernodeoftier}
5479 {\csletcs{forest@tiers@container@forest@tiers@temptier}{forest@tiers@container@#1}}%
5480 }%
5481 }%
5482 }
5483 \def\forest@pack@tiers@computeprocessingorder{%
5484 \def\forest@tiers@processingorder{%
5485 \forestoget{tiers}\forest@tiers@cpo@tierstodo
5486 \safeloop
5487 \ifdefempty\forest@tiers@cpo@tierstodo{\forest@tempfalse}{\forest@temptrue}%
5488 \ifforest@temp
5489 \def\forest@tiers@cpo@tiersremaining{%
5490 \def\forest@tiers@cpo@tiersindependent{%

```

```

5491 \forlistloop\forest@pack@tiers@cpo@forhandlerA\forest@tiers@cpo@tierstodo
5492 \ifdefempty\forest@tiers@cpo@tiersindependent{%
5493 \PackageError{forest}{Circular tiers!}{}}{%
5494 \forlistloop\forest@pack@tiers@cpo@forhandlerB\forest@tiers@cpo@tiersremaining
5495 \let\forest@tiers@cpo@tierstodo\forest@tiers@cpo@tiersremaining
5496 \saferepeat
5497 }
5498 \def\forest@pack@tiers@cpo@forhandlerA#1{%
5499 \ifcsempy\forest@tiers@higher@#1{%
5500 \listadd\forest@tiers@cpo@tiersindependent{#1}%
5501 \listadd\forest@tiers@processingorder{#1}%
5502 }{%
5503 \listadd\forest@tiers@cpo@tiersremaining{#1}%
5504 }%
5505 }
5506 \def\forest@pack@tiers@cpo@forhandlerB#1{%
5507 \def\forest@pack@tiers@cpo@aremainingtier{#1}%
5508 \forlistloop\forest@pack@tiers@cpo@forhandlerC\forest@tiers@cpo@tiersindependent
5509 }
5510 \def\forest@pack@tiers@cpo@forhandlerC#1{%
5511 \ifinlistcs{#1}{\forest@tiers@higher@\forest@pack@tiers@cpo@aremainingtier}{%
5512 \forest@listcsdel{\forest@tiers@higher@\forest@pack@tiers@cpo@aremainingtier}{#1}%
5513 }{}%
5514 }
5515 \def\forest@pack@tiers@write{%
5516 \forlistloop\forest@pack@tiers@write@forhandler\forest@tiers@processingorder
5517 }
5518 \def\forest@pack@tiers@write@forhandler#1{%
5519 \forest@fornode{\csname forest@tiers@container@#1\endcsname}{%
5520 \forest@pack@tiers@check{#1}%
5521 }%
5522 \xappto\forest@smuggle{%
5523 \noexpand\listadd
5524 \forest@om{\csname forest@tiers@container@#1\endcsname}{@tiers}%
5525 {#1}%
5526 }%
5527 }
5528 % checks if the tier is compatible with growth changes and calign=node/edge angle
5529 \def\forest@pack@tiers@check#1{%
5530 \def\forest@temp@currenttier{#1}%
5531 \forest@node@foreachdescendant{%
5532 \ifnum\forestove{grow}=\forestOve{\forestove{@parent}}{grow}
5533 \else
5534 \forest@pack@tiers@check@grow
5535 \fi
5536 \ifnum\forestove{n children}>1
5537 \foresttoget{calign}\forest@temp
5538 \ifx\forest@temp\forest@pack@tiers@check@nodeangle
5539 \forest@pack@tiers@check@calign
5540 \fi
5541 \ifx\forest@temp\forest@pack@tiers@check@edgeangle
5542 \forest@pack@tiers@check@calign
5543 \fi
5544 \fi
5545 }%
5546 }
5547 \def\forest@pack@tiers@check@nodeangle{node angle}%
5548 \def\forest@pack@tiers@check@edgeangle{edge angle}%
5549 \def\forest@pack@tiers@check@grow{%
5550 \foresttoget{content}\forest@temp@content
5551 \let\forest@temp@currentnode\forest@cn

```

```

5552 \forest@node@foreachdescendant{%
5553   \forestoget{tier}\forest@temp
5554   \ifx\forest@temp@currenttier\forest@temp
5555     \forest@pack@tiers@check@grow@error
5556   \fi
5557 }%
5558 }
5559 \def\forest@pack@tiers@check@grow@error{%
5560   \PackageError{forest}{Tree growth direction changes in node \forest@temp@currentnode\space
5561     (content: \forest@temp@content), while tier '\forest@temp' is specified for nodes both
5562     out- and inside the subtree rooted in node \forest@temp@currentnode. This will not work.}{}%
5563 }
5564 \def\forest@pack@tiers@check@calign{%
5565   \forest@node@foreachchild{%
5566     \forestoget{tier}\forest@temp
5567     \ifx\forest@temp@currenttier\forest@temp
5568       \forest@pack@tiers@check@calign@warning
5569     \fi
5570   }%
5571 }
5572 \def\forest@pack@tiers@check@calign@warning{%
5573   \PackageWarning{forest}{Potential option conflict: node \forestove{@parent} (content:
5574     '\forestove{\forestove{@parent}}{content}') was given 'calign=\forestove{calign}', while its
5575     child \forest@cn\space (content: '\forestove{content}') was given 'tier=\forestove{tier}'.
5576     The parent's 'calign' will only work if the child was the lowest node on its tier before the
5577     alignment.}%
5578 }

```

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.

```

5579 \pgfdeclaregenericanchor{forestcomputenodeboundary}{%
5580   \letcs\forest@temp@boundary@macro{forest@compute@node@boundary@#1}%
5581   \ifcsname forest@compute@node@boundary@#1\endcsname
5582     \csname forest@compute@node@boundary@#1\endcsname
5583   \else
5584     \forest@compute@node@boundary@rectangle
5585   \fi
5586   \pgfsyssoftpath@getcurrentpath\forest@temp
5587   \global\let\forest@global@boundary\forest@temp
5588 }
5589 \def\forest@mt#1{%
5590   \expandafter\pgfpointanchor\expandafter{\pgfreferencednodename}{#1}%
5591   \pgfsyssoftpath@moveto{\the\pgf@x}{\the\pgf@y}%
5592 }%
5593 \def\forest@lt#1{%
5594   \expandafter\pgfpointanchor\expandafter{\pgfreferencednodename}{#1}%
5595   \pgfsyssoftpath@lineto{\the\pgf@x}{\the\pgf@y}%
5596 }%
5597 \def\forest@compute@node@boundary@coordinate{%
5598   \forest@mt{center}%
5599 }
5600 \def\forest@compute@node@boundary@circle{%
5601   \forest@mt{east}%
5602   \forest@lt{north east}%
5603   \forest@lt{north}%
5604   \forest@lt{north west}%
5605   \forest@lt{west}%
5606   \forest@lt{south west}%

```

```

5607 \forest@lt{south}%
5608 \forest@lt{south east}%
5609 \forest@lt{east}%
5610 }
5611 \def\forest@compute@node@boundary@rectangle{%
5612 \forest@mt{south west}%
5613 \forest@lt{south east}%
5614 \forest@lt{north east}%
5615 \forest@lt{north west}%
5616 \forest@lt{south west}%
5617 }
5618 \def\forest@compute@node@boundary@diamond{%
5619 \forest@mt{east}%
5620 \forest@lt{north}%
5621 \forest@lt{west}%
5622 \forest@lt{south}%
5623 \forest@lt{east}%
5624 }
5625 \let\forest@compute@node@boundary@ellipse\forest@compute@node@boundary@circle
5626 \def\forest@compute@node@boundary@trapezium{%
5627 \forest@mt{top right corner}%
5628 \forest@lt{top left corner}%
5629 \forest@lt{bottom left corner}%
5630 \forest@lt{bottom right corner}%
5631 \forest@lt{top right corner}%
5632 }
5633 \def\forest@compute@node@boundary@semicircle{%
5634 \forest@mt{arc start}%
5635 \forest@lt{north}%
5636 \forest@lt{east}%
5637 \forest@lt{north east}%
5638 \forest@lt{apex}%
5639 \forest@lt{north west}%
5640 \forest@lt{west}%
5641 \forest@lt{arc end}%
5642 \forest@lt{arc start}%
5643 }
5644 %\newloop\forest@computenodeboundary@loop
5645 \csdef{forest@compute@node@boundary@regular polygon}{%
5646 \forest@mt{corner 1}%
5647 \c@pgf@counta=\sides\relax
5648 \forest@loop
5649 \ifnum\c@pgf@counta>0
5650 \forest@lt{corner \the\c@pgf@counta}%
5651 \advance\c@pgf@counta-1
5652 \forest@repeat
5653 }%
5654 \def\forest@compute@node@boundary@star{%
5655 \forest@mt{outer point 1}%
5656 \c@pgf@counta=\totalstarpoints\relax
5657 \divide\c@pgf@counta2
5658 \forest@loop
5659 \ifnum\c@pgf@counta>0
5660 \forest@lt{inner point \the\c@pgf@counta}%
5661 \forest@lt{outer point \the\c@pgf@counta}%
5662 \advance\c@pgf@counta-1
5663 \forest@repeat
5664 }%
5665 \csdef{forest@compute@node@boundary@isosceles triangle}{%
5666 \forest@mt{apex}%
5667 \forest@lt{left corner}%

```

```

5668 \forest@lt{right corner}%
5669 \forest@lt{apex}%
5670 }
5671 \def\forest@compute@node@boundary@kite{%
5672 \forest@mt{upper vertex}%
5673 \forest@lt{left vertex}%
5674 \forest@lt{lower vertex}%
5675 \forest@lt{right vertex}%
5676 \forest@lt{upper vertex}%
5677 }
5678 \def\forest@compute@node@boundary@dart{%
5679 \forest@mt{tip}%
5680 \forest@lt{left tail}%
5681 \forest@lt{tail center}%
5682 \forest@lt{right tail}%
5683 \forest@lt{tip}%
5684 }
5685 \csdef{forest@compute@node@boundary@circular sector}{%
5686 \forest@mt{sector center}%
5687 \forest@lt{arc start}%
5688 \forest@lt{arc center}%
5689 \forest@lt{arc end}%
5690 \forest@lt{sector center}%
5691 }
5692 \def\forest@compute@node@boundary@cylinder{%
5693 \forest@mt{top}%
5694 \forest@lt{after top}%
5695 \forest@lt{before bottom}%
5696 \forest@lt{bottom}%
5697 \forest@lt{after bottom}%
5698 \forest@lt{before top}%
5699 \forest@lt{top}%
5700 }
5701 \cslet{forest@compute@node@boundary@forbidden sign}\forest@compute@node@boundary@circle
5702 \cslet{forest@compute@node@boundary@magnifying glass}\forest@compute@node@boundary@circle
5703 \def\forest@compute@node@boundary@cloud{%
5704 \getradii
5705 \forest@mt{puff 1}%
5706 \c@pgf@counta=\puffs\relax
5707 \forest@loop
5708 \ifnum\c@pgf@counta>0
5709 \forest@lt{puff \the\c@pgf@counta}%
5710 \advance\c@pgf@counta-1
5711 \forest@repeat
5712 }
5713 \def\forest@compute@node@boundary@starburst{
5714 \calculatestarburstpoints
5715 \forest@mt{outer point 1}%
5716 \c@pgf@counta=\totalpoints\relax
5717 \divide\c@pgf@counta2
5718 \forest@loop
5719 \ifnum\c@pgf@counta>0
5720 \forest@lt{inner point \the\c@pgf@counta}%
5721 \forest@lt{outer point \the\c@pgf@counta}%
5722 \advance\c@pgf@counta-1
5723 \forest@repeat
5724 }%
5725 \def\forest@compute@node@boundary@signal{%
5726 \forest@mt{east}%
5727 \forest@lt{south east}%
5728 \forest@lt{south west}%

```

```

5729 \forest@lt{west}%
5730 \forest@lt{north west}%
5731 \forest@lt{north east}%
5732 \forest@lt{east}%
5733 }
5734 \def\forest@compute@node@boundary@tape{%
5735 \forest@mt{north east}%
5736 \forest@lt{60}%
5737 \forest@lt{north}%
5738 \forest@lt{120}%
5739 \forest@lt{north west}%
5740 \forest@lt{south west}%
5741 \forest@lt{240}%
5742 \forest@lt{south}%
5743 \forest@lt{310}%
5744 \forest@lt{south east}%
5745 \forest@lt{north east}%
5746 }
5747 \csdef{forest@compute@node@boundary@single arrow}{%
5748 \forest@mt{tip}%
5749 \forest@lt{after tip}%
5750 \forest@lt{after head}%
5751 \forest@lt{before tail}%
5752 \forest@lt{after tail}%
5753 \forest@lt{before head}%
5754 \forest@lt{before tip}%
5755 \forest@lt{tip}%
5756 }
5757 \csdef{forest@compute@node@boundary@double arrow}{%
5758 \forest@mt{tip 1}%
5759 \forest@lt{after tip 1}%
5760 \forest@lt{after head 1}%
5761 \forest@lt{before head 2}%
5762 \forest@lt{before tip 2}%
5763 \forest@mt{tip 2}%
5764 \forest@lt{after tip 2}%
5765 \forest@lt{after head 2}%
5766 \forest@lt{before head 1}%
5767 \forest@lt{before tip 1}%
5768 \forest@lt{tip 1}%
5769 }
5770 \csdef{forest@compute@node@boundary@arrow box}{%
5771 \forest@mt{before north arrow}%
5772 \forest@lt{before north arrow head}%
5773 \forest@lt{before north arrow tip}%
5774 \forest@lt{north arrow tip}%
5775 \forest@lt{after north arrow tip}%
5776 \forest@lt{after north arrow head}%
5777 \forest@lt{after north arrow}%
5778 \forest@lt{north east}%
5779 \forest@lt{before east arrow}%
5780 \forest@lt{before east arrow head}%
5781 \forest@lt{before east arrow tip}%
5782 \forest@lt{east arrow tip}%
5783 \forest@lt{after east arrow tip}%
5784 \forest@lt{after east arrow head}%
5785 \forest@lt{after east arrow}%
5786 \forest@lt{south east}%
5787 \forest@lt{before south arrow}%
5788 \forest@lt{before south arrow head}%
5789 \forest@lt{before south arrow tip}%

```

```

5790 \forest@lt{south arrow tip}%
5791 \forest@lt{after south arrow tip}%
5792 \forest@lt{after south arrow head}%
5793 \forest@lt{after south arrow}%
5794 \forest@lt{south west}%
5795 \forest@lt{before west arrow}%
5796 \forest@lt{before west arrow head}%
5797 \forest@lt{before west arrow tip}%
5798 \forest@lt{west arrow tip}%
5799 \forest@lt{after west arrow tip}%
5800 \forest@lt{after west arrow head}%
5801 \forest@lt{after west arrow}%
5802 \forest@lt{north west}%
5803 \forest@lt{before north arrow}%
5804 }
5805 \cslet{forest@compute@node@boundary@circle split}\forest@compute@node@boundary@circle
5806 \cslet{forest@compute@node@boundary@circle solidus}\forest@compute@node@boundary@circle
5807 \cslet{forest@compute@node@boundary@ellipse split}\forest@compute@node@boundary@ellipse
5808 \cslet{forest@compute@node@boundary@rectangle split}\forest@compute@node@boundary@rectangle
5809 \def\forest@compute@node@boundary@@callout{%
5810 \beforecalloutpointer
5811 \pgfsyssoftpath@moveto{\the\pgf@x}{\the\pgf@y}%
5812 \calloutpointeranchor
5813 \pgfsyssoftpath@lineto{\the\pgf@x}{\the\pgf@y}%
5814 \aftercalloutpointer
5815 \pgfsyssoftpath@lineto{\the\pgf@x}{\the\pgf@y}%
5816 }
5817 \csdef{forest@compute@node@boundary@rectangle callout}{%
5818 \forest@compute@node@boundary@rectangle
5819 \rectanglecalloutpoints
5820 \forest@compute@node@boundary@@callout
5821 }
5822 \csdef{forest@compute@node@boundary@ellipse callout}{%
5823 \forest@compute@node@boundary@ellipse
5824 \ellipsecalloutpoints
5825 \forest@compute@node@boundary@@callout
5826 }
5827 \csdef{forest@compute@node@boundary@cloud callout}{%
5828 \forest@compute@node@boundary@cloud
5829 % at least a first approx...
5830 \forest@mt{center}%
5831 \forest@lt{pointer}%
5832 }%
5833 \csdef{forest@compute@node@boundary@cross out}{%
5834 \forest@mt{south east}%
5835 \forest@lt{north west}%
5836 \forest@mt{south west}%
5837 \forest@lt{north east}%
5838 }%
5839 \csdef{forest@compute@node@boundary@strike out}{%
5840 \forest@mt{north east}%
5841 \forest@lt{south west}%
5842 }%
5843 \csdef{forest@compute@node@boundary@rounded rectangle}{%
5844 \forest@mt{east}%
5845 \forest@lt{north east}%
5846 \forest@lt{north}%
5847 \forest@lt{north west}%
5848 \forest@lt{west}%
5849 \forest@lt{south west}%
5850 \forest@lt{south}%

```



```

5851 \forest@lt{south east}%
5852 \forest@lt{east}%
5853 }%
5854 \csdef{forest@compute@node@boundary@chamfered rectangle}{%
5855 \forest@mt{before south west}%
5856 \forest@mt{after south west}%
5857 \forest@lt{before south east}%
5858 \forest@lt{after south east}%
5859 \forest@lt{before north east}%
5860 \forest@lt{after north east}%
5861 \forest@lt{before north west}%
5862 \forest@lt{after north west}%
5863 \forest@lt{before south west}%
5864 }%

```

7.3 Compute absolute positions

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

```

5865 \def\forest@node@computeabsolutepositions{%
5866 \edef\forest@marshal{%
5867 \noexpand\forest@node@foreachchild{%
5868 \noexpand\forest@node@computeabsolutepositions@{\forestove{x}}{\forestove{y}}{\forestove{grow}}}%
5869 }%
5870 }\forest@marshal
5871 }
5872 \def\forest@node@computeabsolutepositions@#1#2#3{%
5873 \pgfpointadd{\pgfpoint{#1}{#2}}{%
5874 \pgfpointadd{\pgfpolar{#3}{\forestove{1}}}{\pgfpolar{90 + #3}{\forestove{s}}}}}%
5875 \pgfgetlastxy\forest@tempx\forest@tempy
5876 \forestolet{x}\forest@tempx
5877 \forestolet{y}\forest@tempy
5878 \edef\forest@marshal{%
5879 \noexpand\forest@node@foreachchild{%
5880 \noexpand\forest@node@computeabsolutepositions@{\forest@tempx}{\forest@tempy}{\forestove{grow}}}%
5881 }%
5882 }\forest@marshal
5883 }

```

7.4 Drawing the tree

```

5884 \newif\ifforest@drawtree@preservenodeboxes@
5885 \def\forest@node@drawtree{%
5886 \expandafter\ifstrequal\expandafter{\forest@drawtreebox}{\pgfkeysnovalue}{%
5887 \let\forest@drawtree@beginbox\relax
5888 \let\forest@drawtree@endbox\relax
5889 }{%
5890 \edef\forest@drawtree@beginbox{\global\setbox\forest@drawtreebox=\hbox\bgroup}%
5891 \let\forest@drawtree@endbox\egroup
5892 }%
5893 \ifforest@external@
5894 \ifforest@externalize@tree@
5895 \forest@temptrue
5896 \else
5897 \tikzifexternalizing{%
5898 \ifforest@was@tikzexternalwasenable
5899 \forest@temptrue
5900 \pgfkeys{/tikz/external/optimize=false}%
5901 \let\forest@drawtree@beginbox\relax
5902 \let\forest@drawtree@endbox\relax

```

```

5903     \else
5904     \forest@tempfalse
5905     \fi
5906   }{%
5907     \forest@tempfalse
5908   }%
5909 \fi
5910 \ifforest@temp
5911   \advance\forest@externalize@inner@n 1
5912   \edef\forest@externalize@filename{%
5913     \tikzexternalrealjob-forest-\forest@externalize@outer@n
5914     \ifnum\forest@externalize@inner@n=0 \else.\the\forest@externalize@inner@n\fi}%
5915   \expandafter\tikzsetnextfilename\expandafter{\forest@externalize@filename}%
5916   \tikzexternalenable
5917   \pgfkeysalso{/tikz/external/remake next,/tikz/external/export next}%
5918 \fi
5919 \ifforest@externalize@tree@
5920   \typeout{forest: Invoking a recursive call to generate the external picture
5921     '\forest@externalize@filename' for the following context+code:
5922     '\expandafter\detokenize\expandafter{\forest@externalize@id}'}%
5923 \fi
5924 \fi
5925 %
5926 \ifforest\tikzcshack
5927   \let\forest@original@tikz@parse@node\tikz@parse@node
5928   \let\tikz@parse@node\forest@tikz@parse@node
5929 \fi
5930 \pgfkeysgetvalue{/forest/begin draw/.@cmd}\forest@temp@begindraw
5931 \pgfkeysgetvalue{/forest/end draw/.@cmd}\forest@temp@enddraw
5932 \edef\forest@marshal{%
5933   \noexpand\forest@drawtree@beginbox
5934   \expandonce{\forest@temp@begindraw\pgfkeysnovalue\pgfeov}%
5935   \noexpand\forest@node@drawtree@
5936   \expandonce{\forest@temp@enddraw\pgfkeysnovalue\pgfeov}%
5937   \noexpand\forest@drawtree@endbox
5938 }%
5939 \ifforest\tikzcshack
5940   \let\tikz@parse@node\forest@original@tikz@parse@node
5941 \fi
5942 %
5943 \ifforest@external@
5944   \ifforest@externalize@tree@
5945     \tikzexternaldisable
5946     \eappto\forest@externalize@checkimages{%
5947       \noexpand\forest@includeexternal@check{\forest@externalize@filename}%
5948     }%
5949     \expandafter\ifstrequal\expandafter{\forest@drawtreebox}{\pgfkeysnovalue}{%
5950       \eappto\forest@externalize@loadimages{%
5951         \noexpand\forest@includeexternal{\forest@externalize@filename}%
5952       }%
5953     }{%
5954       \eappto\forest@externalize@loadimages{%
5955         \noexpand\forest@includeexternal@box\forest@drawtreebox{\forest@externalize@filename}%
5956       }%
5957     }%
5958   \fi
5959 \fi
5960 }
5961 \def\forest@drawtree@root{0}
5962 \def\forest@node@drawtree{%
5963   \def\forest@clear@drawn{}%

```

```

5964 \forest@forthis{%
5965   \forest@saveandrestoremacro\forest@drawtree@root{%
5966     \edef\forest@drawtree@root{\forest@cn}%
5967     \forestset{draw tree method}%
5968   }%
5969 }%
5970 \forest@node@ifnamedefined{forest@baseline@node}{%
5971   \edef\forest@baseline@id{\forest@node@Nametoid{forest@baseline@node}}%
5972   \ifcsdef{forest@drawn@\forest@baseline@id}{%
5973     \edef\forest@marshal{%
5974       \noexpand\pgfsetbaselinepointlater{%
5975         \noexpand\pgfpntanchor
5976         {\forestOve{\forest@baseline@id}{name}}%
5977         {\forestOve{\forest@baseline@id}{anchor}}%
5978       }%
5979     }\forest@marshal
5980   }{}%
5981 }{}%
5982 \forest@clear@drawn
5983 }
5984 \def\forest@draw@node{%
5985   \ifnum\forestove{phantom}=0
5986     \forest@draw@node@
5987   \fi
5988 }
5989 \def\forest@draw@node@{%
5990   \forest@node@forest@position@node@later@restore
5991   \ifforest@drawtree@preservenodeboxes@
5992     \pgfnodealias{forest@temp}{\forestove{later@name}}%
5993   \fi
5994   \pgfpositionnodenow{\pgfpnt{\forestove{x}}{\forestove{y}}}%
5995   \ifforest@drawtree@preservenodeboxes@
5996     \pgfnodealias{\forestove{later@name}}{forest@temp}%
5997   \fi
5998   \csdef{forest@drawn@\forest@cn}{}%
5999   \eappto\forest@clear@drawn{\noexpand\csundef{forest@drawn@\forest@cn}}%
6000 }
6001 \def\forest@draw@edge{%
6002   \ifcsdef{forest@drawn@\forest@cn}{% was the current node drawn?
6003     \ifnum\forestove{@parent}=0 % do we have a parent?
6004       \else
6005         \ifcsdef{forest@drawn@\forestove{@parent}}{% was the parent drawn?
6006           \forest@draw@edge@
6007         }{}%
6008       \fi
6009     }{}%
6010 }
6011 \def\forest@draw@edge@{%
6012   \edef\forest@temp{\forestove{edge path}}\forest@temp
6013 }
6014 \def\forest@draw@tikz{%
6015   \ifnum\forestove{phantom}=0
6016     \forest@draw@tikz@
6017   \fi
6018 }
6019 \def\forest@draw@tikz@{%
6020   \forestove{tikz}%
6021 }

```

8 Geometry

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

```

6022 \newdimen\forest@xg
6023 \newdimen\forest@yg
6024 \newdimen\forest@xs
6025 \newdimen\forest@ys
6026 \def\forest@setupgrowline#1{%
6027   \edef\forest@grow{#1}%
6028   \pgfpointpolar\forest@grow{1pt}%
6029   \forest@xg=\pgf@x
6030   \forest@yg=\pgf@y
6031   \forest@xs=-\pgf@y
6032   \forest@ys=\pgf@x
6033 }
```

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`.

```

6034 \def\forest@pgfpointprojectiontogrowline#1{%
6035   \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.
6036   \pgfutil@tempdima=\pgf@sys@tonumber{\pgf@x}\forest@xg%
6037   \advance\pgfutil@tempdima by\pgf@sys@tonumber{\pgf@y}\forest@yg%
```

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

```

6038   \global\pgf@x=\pgf@sys@tonumber{\pgfutil@tempdima}\forest@xg%
6039   \global\pgf@y=\pgf@sys@tonumber{\pgfutil@tempdima}\forest@yg%
6040 }
```

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.

```

6041 \def\forest@distancetogrowline#1#2{%
6042   \pgf@process{#2}%
6043   #1=\pgf@sys@tonumber{\pgf@x}\forest@xs\relax
6044   \advance#1 by\pgf@sys@tonumber{\pgf@y}\forest@ys\relax
6045 }
```

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`.

```

6046 \let\forest@pp@n\relax
6047 \def\forest@projectpathtogrowline#1#2{%
6048   \edef\forest@pp@prefix{#2}%
6049   \forest@save@pgfsyssoftpath@tokendefs
```

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

```

6050 \let\pgfsyssoftpath@movetotoken\forest@projectpath@processpoint
6051 \let\pgfsyssoftpath@linetotoken\forest@projectpath@processpoint
6052 \c@pgf@counta=0
6053 #1%
6054 \csedef{#2n}{\the\c@pgf@counta}%
6055 \forest@restore@pgfsyssoftpath@tokendefs
6056 }

```

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

```

6057 \def\forest@projectpath@processpoint#1#2{%
6058   \pgfqpoint{#1}{#2}%
6059   \expandafter\edef\csname\forest@pp@prefix\the\c@pgf@counta xo\endcsname{\the\pgf@x}%
6060   \expandafter\edef\csname\forest@pp@prefix\the\c@pgf@counta yo\endcsname{\the\pgf@y}%
6061   \forest@pgfpointprojectiontogrowline{}%
6062   \expandafter\edef\csname\forest@pp@prefix\the\c@pgf@counta xp\endcsname{\the\pgf@x}%
6063   \expandafter\edef\csname\forest@pp@prefix\the\c@pgf@counta yp\endcsname{\the\pgf@y}%
6064   \advance\c@pgf@counta 1\relax
6065 }

```

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

```

6066 \def\forest@sortprojections#1{%
6067   % todo: optimize in cases when we know that the array is actually a
6068   % merger of sorted arrays; when does this happen? in
6069   % distance_between_paths, and when merging the edges of the parent
6070   % and its children in a uniform growth tree
6071   \edef\forest@ppi@inputprefix{#1}%
6072   \c@pgf@counta=\csname#1n\endcsname\relax
6073   \advance\c@pgf@counta -1
6074   \forest@sort\forest@ppiraw@cmp\forest@ppiraw@let\forest@sort@ascending{0}{\the\c@pgf@counta}%
6075 }

```

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

```

6076 \def\forest@processprojectioninfo#1#2{%
6077   \edef\forest@ppi@inputprefix{#1}%

```

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

```

6078 \c@pgf@counta=0
6079 \c@pgf@countb=-1
6080 \safeloop
6081 \ifnum\c@pgf@counta<\csname#1n\endcsname\relax

```

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

```

6082 \letcs\forest@xo{#1\the\c@pgf@counta xo}%
6083 \letcs\forest@yo{#1\the\c@pgf@counta yo}%
6084 \letcs\forest@xp{#1\the\c@pgf@counta xp}%
6085 \letcs\forest@yp{#1\the\c@pgf@counta yp}%

```

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.

```

6086 \ifnum\c@pgf@countb<0
6087 \forest@equaltotolerancefalse
6088 \else
6089 \forest@equaltotolerance
6090 {\pgfqpoint\forest@xp\forest@yp}%
6091 {\pgfqpoint
6092 {\csname#2\the\c@pgf@countb x\endcsname}%
6093 {\csname#2\the\c@pgf@countb y\endcsname}%
6094 }%
6095 \fi
6096 \ifforest@equaltotolerance\else

```

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

```

6097 \advance\c@pgf@countb 1
6098 \cslet{#2\the\c@pgf@countb x}\forest@xp
6099 \cslet{#2\the\c@pgf@countb y}\forest@yp
6100 \csdef{#2\the\c@pgf@countb @n}{0}%
6101 \fi

```

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

```

6102 % todo: this is ugly!
6103 \ifdefined\forest@xo\ifx\forest@xo\relax\else
6104 \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.

```

6105 \forest@append@point@to@inner@array
6106 \forest@xo\forest@yo
6107 {#2\the\c@pgf@countb @}%

```

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

```

6108 \csedef{#2(\forest@xo,\forest@yo)}{\the\c@pgf@countb}%
6109 \fi\fi
6110 \fi\fi

```

Clean-up the raw array item.

```

6111 \cslet{#1\the\c@pgf@counta xo}\relax
6112 \cslet{#1\the\c@pgf@counta yo}\relax
6113 \cslet{#1\the\c@pgf@counta xp}\relax
6114 \cslet{#1\the\c@pgf@counta yp}\relax
6115 \advance\c@pgf@counta 1
6116 \saferepeat

```

Clean up the raw array length.

```

6117 \cslet{#1n}\relax

```

Store the length of the outer result array.

```

6118 \advance\c@pgf@countb 1
6119 \csedef{#2n}{\the\c@pgf@countb}%
6120 }

```

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

```

6121 \def\forest@ppiraw@let#1#2{%
6122 \csletcs{\forest@ppi@inputprefix#1xo}{\forest@ppi@inputprefix#2xo}%
6123 \csletcs{\forest@ppi@inputprefix#1yo}{\forest@ppi@inputprefix#2yo}%
6124 \csletcs{\forest@ppi@inputprefix#1xp}{\forest@ppi@inputprefix#2xp}%
6125 \csletcs{\forest@ppi@inputprefix#1yp}{\forest@ppi@inputprefix#2yp}%
6126 }

```

Item comparison macro for quicksorting the raw projection data.

```

6127 \def\forest@ppiraw@cmp#1#2{%
6128 \forest@sort@cmptwodimcs
6129 {\forest@ppi@inputprefix#1xp}{\forest@ppi@inputprefix#1yp}%
6130 {\forest@ppi@inputprefix#2xp}{\forest@ppi@inputprefix#2yp}%
6131 }

```

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

```

6132 \def\forest@append@point@to@inner@array#1#2#3{%
6133   \c@pgf@countc=\csname#3n\endcsname\relax
6134   \csedef{#3\the\c@pgf@countc x}{#1}%
6135   \csedef{#3\the\c@pgf@countc y}{#2}%
6136   \forest@distancetogrowline\pgfutil@tempdima{\pgfqpoint{#1}{#2}}%
6137   \csedef{#3\the\c@pgf@countc d}{\the\pgfutil@tempdima}%
6138   \advance\c@pgf@countc 1
6139   \csedef{#3n}{\the\c@pgf@countc}%
6140 }

```

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.

```

6141 \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.
6142   \edef\forest@bp@prefix{#2}%
6143   \forest@save\pgfsyssoftpath@tokendefs
6144   \let\pgfsyssoftpath@movetotoken\forest@breakpath@processfirstpoint
6145   \let\pgfsyssoftpath@linetotoken\forest@breakpath@processfirstpoint
6146   %\pgfusepath{}% empty the current path. ok?
6147   #1%
6148   \forest@restore\pgfsyssoftpath@tokendefs
6149   \pgfsyssoftpath@getcurrentpath#3%
6150 }

```

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.)

```

6151 \def\forest@breakpath@processfirstpoint#1#2{%
6152   \forest@breakpath@processmoveto{#1}{#2}%
6153   \let\pgfsyssoftpath@movetotoken\forest@breakpath@processmoveto
6154   \let\pgfsyssoftpath@linetotoken\forest@breakpath@processlineto
6155 }

```

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.

```

6156 \def\forest@breakpath@processmoveto#1#2{%
6157   \pgfsyssoftpath@moveto{#1}{#2}%
6158   \def\forest@previous{x{#1}}%
6159   \def\forest@previous{y{#2}}%
6160   \expandafter\let\expandafter\forest@previous@i
6161     \csname\forest@bp@prefix(#1,#2)\endcsname
6162   \expandafter\let\expandafter\forest@previous@px
6163     \csname\forest@bp@prefix\forest@previous@i x\endcsname
6164   \expandafter\let\expandafter\forest@previous@py
6165     \csname\forest@bp@prefix\forest@previous@i y\endcsname
6166 }

```

This is the heart of the path-breaking procedure.

```

6167 \def\forest@breakpath@processlineto#1#2{%

```

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

```
6168 \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.)

```
6169 \expandafter\let\expandafter\forest@i
6170 \csname\forest@bp@prefix(#1,#2)\endcsname
6171 \expandafter\let\expandafter\forest@px
6172 \csname\forest@bp@prefix\forest@i x\endcsname
6173 \expandafter\let\expandafter\forest@py
6174 \csname\forest@bp@prefix\forest@i y\endcsname
```

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

```
6175 \forest@equaltotolerance
6176 {\pgfqpoint{\forest@previous@px}{\forest@previous@py}}%
6177 {\pgfqpoint{\forest@px}{\forest@py}}%
6178 \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.

```
6179 \let\forest@breakpath@op\pgfsyssoftpath@moveto
6180 \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.

```
6181 \forest@temp@count=\forest@previous@i\relax
6182 \ifnum\forest@previous@i<\forest@i\relax
6183 \def\forest@breakpath@step{1}%
6184 \def\forest@breakpath@test{\forest@temp@count<\forest@i\relax}%
6185 \else
6186 \def\forest@breakpath@step{-1}%
6187 \def\forest@breakpath@test{\forest@temp@count>\forest@i\relax}%
6188 \fi
```

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

```
6189 \safeloop
6190 \advance\forest@temp@count\forest@breakpath@step\relax
6191 \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.)

```
6192 \pgfpointintersectionoflines
6193 {\pgfqpoint
6194 {\csname\forest@bp@prefix\the\forest@temp@count x\endcsname}%
6195 {\csname\forest@bp@prefix\the\forest@temp@count y\endcsname}%
6196 }%
6197 {\pgfpointadd
6198 {\pgfqpoint
6199 {\csname\forest@bp@prefix\the\forest@temp@count x\endcsname}%
6200 {\csname\forest@bp@prefix\the\forest@temp@count y\endcsname}%
6201 }%
6202 {\pgfqpoint{\forest@xs}{\forest@ys}}%
6203 }%
6204 {\pgfqpoint{\forest@previous@x}{\forest@previous@y}}%
6205 {\pgfqpoint{#1}{#2}}%
```

Break the segment at the intersection.

```
6206 \pgfgetlastxy\forest@last@x\forest@last@y
6207 \pgfsyssoftpath@lineto\forest@last@x\forest@last@y
```

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

```
6208 \forest@append@point@to@inner@array
```



```

6209      \forest@last@x\forest@last@y
6210      {\forest@bp@prefix\the\forest@temp@count @}%
Cache the projection of the new segment edge.
6211      \csedef{\forest@bp@prefix(\the\pgf@x,\the\pgf@y)}{\the\forest@temp@count}%
6212      \saferepeat
6213      \fi
Add the current point.
6214      \forest@breakpath@op{#1}{#2}%
Setup new “previous” info: the segment edge, its projection’s index, and the projection.
6215      \def\forest@previous@x{#1}%
6216      \def\forest@previous@y{#2}%
6217      \let\forest@previous@i\forest@i
6218      \let\forest@previous@px\forest@px
6219      \let\forest@previous@py\forest@py
6220 }

```

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.

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.

```

6221 \def\forest@getnegativetightedgeofpath#1#2{%
6222   \forest@get@onetightedgeofpath#1\forest@sort@ascending#2}
6223 \def\forest@getpositivetightedgeofpath#1#2{%
6224   \forest@get@onetightedgeofpath#1\forest@sort@descending#2}
6225 \def\forest@get@onetightedgeofpath#1#2#3{%
6226   {%
6227     \forest@get@one@tightedgeofpath#1#2\forest@gep@edge
6228     \global\let\forest@gep@global@edge\forest@gep@edge
6229   }%
6230   \let#3\forest@gep@global@edge
6231 }
6232 \def\forest@get@one@tightedgeofpath#1#2#3{%

```

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

```

6233   \forest@projectpathtogrowline#1\forest@pp@}%
6234   \forest@sortprojections\forest@pp@}%
6235   \forest@processprojectioninfo\forest@pp@}\forest@pi@}%

```

Break the path.

```

6236   \forest@breakpath#1\forest@pi@}\forest@brokenpath

```

Compile some more useful information.

```

6237   \forest@sort@inner@arrays\forest@pi@}\forest@pi@}%
6238   \forest@pathtodict\forest@brokenpath\forest@pi@}%

```

The auxiliary data is set up: do the work!

```

6239   \forest@gettightedgeofpath@getedge\forest@edge

```

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

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).

```
6240 \forest@simplifypath\forest@edge#3%
6241 }
```

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

```
6242 \def\forest@getbothtightedgesofpath#1#2#3{%
6243   {%
6244     \forest@get@one@tightedgeofpath#1\forest@sort@ascending\forest@gep@firstedge
```

Reverse the order of items in the inner arrays.

```
6245     \c@pgf@counta=0
6246     \forest@loop
6247     \ifnum\c@pgf@counta<\forest@pi@n\relax
6248       \forest@ppi@deflet{\forest@pi@the\c@pgf@counta @}%
6249       \forest@reversearray\forest@ppi@let
6250       {0}%
6251       {\csname forest@pi@the\c@pgf@counta @n\endcsname}%
6252       \advance\c@pgf@counta 1
6253     \forest@repeat
```

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

```
6254     \forest@gettightedgeofpath@getedge\forest@edge
6255     \forest@simplifypath\forest@edge\forest@gep@secondedge
```

Smuggle the results out of the enclosing TeX group.

```
6256     \global\let\forest@gep@global@firstedge\forest@gep@firstedge
6257     \global\let\forest@gep@global@secondedge\forest@gep@secondedge
6258   }%
6259   \let#2\forest@gep@global@firstedge
6260   \let#3\forest@gep@global@secondedge
6261 }
```

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

```
6262 \def\forest@sort@inner@arrays#1#2{%
6263   \c@pgf@counta=0
6264   \safeloop
6265   \ifnum\c@pgf@counta<\csname#1n\endcsname
6266     \c@pgf@countb=\csname#1the\c@pgf@counta @n\endcsname\relax
6267     \ifnum\c@pgf@countb>1
6268       \advance\c@pgf@countb -1
6269       \forest@ppi@deflet{#1the\c@pgf@counta @}%
6270       \forest@ppi@defcmp{#1the\c@pgf@counta @}%
6271       \forest@sort\forest@ppi@cmp\forest@ppi@let#2{0}{\the\c@pgf@countb}%
6272     \fi
6273     \advance\c@pgf@counta 1
6274   \saferepeat
6275 }
```

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.

```
6276 \def\forest@ppi@deflet#1{%
6277   \edef\forest@ppi@let##1##2{%
6278     \noexpand\csletcs{#1##1x}{#1##2x}%
6279     \noexpand\csletcs{#1##1y}{#1##2y}%
6280     \noexpand\csletcs{#1##1d}{#1##2d}%
6281   }%
6282 }
```

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.

```
6283 \def\forest@ppi@defcmp#1{%
```

```

6284 \edef\forest@ppi@cmp##1##2{%
6285   \noexpand\forest@sort@cmpdimcs{#1##1d}{#1##2d}%
6286 }%
6287 }

```

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

```

6288 \let\forest@inpath\advance

```

This macro is just a wrapper to process the path.

```

6289 \def\forest@pathtodict#1#2{%
6290   \edef\forest@pathtodict@prefix{#2}%
6291   \forest@save@pgfsyssoftpath@tokendefs
6292   \let\pgfsyssoftpath@movetotoken\forest@pathtodict@movetoop
6293   \let\pgfsyssoftpath@linetotoken\forest@pathtodict@linetoop
6294   \def\forest@pathtodict@subpathstart{}%
6295   #1%
6296   \forest@restore@pgfsyssoftpath@tokendefs
6297 }

```

When a move-to operation is encountered:

```

6298 \def\forest@pathtodict@movetoop#1#2{%

```

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

```

6299   \def\forest@pathtodict@subpathstart{(#1,#2)-}%
6300 }

```

When a line-to operation is encountered:

```

6301 \def\forest@pathtodict@linetoop#1#2{%

```

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

```

6302 \if\relax\forest@pathtodict@subpathstart\relax\else
6303   \let\forest@pathtodict@from\forest@pathtodict@subpathstart
6304 \fi

```

Mark the segment as existing.

```

6305   \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.

```

6306   \def\forest@pathtodict@from{(#1,#2)-}%
6307   \def\forest@pathtodict@subpathstart{}%
6308 }

```

In this macro, the edge is actually computed.

```

6309 \def\forest@gettightedgeofpath@getedge#1{% cs to store the edge into

```

Clear the path and the last projection.

```

6310   \pgfsyssoftpath@setcurrentpath\pgfutil@empty
6311   \let\forest@last@x\relax
6312   \let\forest@last@y\relax

```

Loop through the (ordered) array of projections. (Since we will be dealing with the current and the next projection in each iteration of the loop, we loop the counter from the first to the second-to-last projection.)

```

6313   \c@pgf@counta=0
6314   \forest@temp@count=\forest@pi@n\relax
6315   \advance\forest@temp@count -1
6316   \edef\forest@nminusone{\the\forest@temp@count}%
6317   \safeloop
6318   \ifnum\c@pgf@counta<\forest@nminusone\relax
6319     \forest@gettightedgeofpath@getedge@loopa
6320   \saferepeat

```

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

```

6321 \ifnum\forest@nminusone<\forest@n\relax\else
6322   \ifnum\csname forest@pi@\forest@nminusone @n\endcsname>0
6323     \forest@gettightedgeofpath@maybemoveto{\forest@nminusone}{0}%
6324   \fi
6325 \fi
6326 \pgfsyssoftpath@getcurrentpath#1%
6327 \pgfsyssoftpath@setcurrentpath\pgfutil@empty
6328 }

```

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.

```

6329 \def\forest@gettightedgeofpath@getedge@loopa{%
6330   \ifnum\csname forest@pi@\the\c@pgf@counta @n\endcsname>0

```

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

```

6331   \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.

```

6332   \c@pgf@countb=0
6333   \safeloop
6334   \ifnum\c@pgf@countb<\csname forest@pi@\the\c@pgf@counta @n\endcsname\relax
6335     \forest@gettightedgeofpath@getedge@loopb
6336   \saferepeat
6337 \fi
6338 \advance\c@pgf@counta 1
6339 }

```

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

```

6340 \def\forest@gettightedgeofpath@getedge@loopb{%
6341   \c@pgf@countc=0
6342   \advance\c@pgf@counta 1
6343   \edef\forest@aplusone{\the\c@pgf@counta}%
6344   \advance\c@pgf@counta -1
6345   \safeloop
6346   \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!

```

6347   \forest@tempfalse
6348   \expandafter\ifx\csname forest@pi@(%
6349     \csname forest@pi@\the\c@pgf@counta @\the\c@pgf@countb x\endcsname,%
6350     \csname forest@pi@\the\c@pgf@counta @\the\c@pgf@countb y\endcsname)--(%
6351     \csname forest@pi@\forest@aplusone @\the\c@pgf@countc x\endcsname,%
6352     \csname forest@pi@\forest@aplusone @\the\c@pgf@countc y\endcsname)%
6353   \endcsname\forest@inpath
6354   \forest@temptrue
6355 \else
6356   \expandafter\ifx\csname forest@pi@(%
6357     \csname forest@pi@\forest@aplusone @\the\c@pgf@countc x\endcsname,%
6358     \csname forest@pi@\forest@aplusone @\the\c@pgf@countc y\endcsname)--(%

```

```

6359         \csname forest@pi@the\c@pgf@counta @the\c@pgf@countb x\endcsname,%
6360         \csname forest@pi@the\c@pgf@counta @the\c@pgf@countb y\endcsname)%
6361         \endcsname\forest@inpath
6362         \forest@temptrue
6363     \fi
6364 \fi
6365 \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.

```

6366     \forest@gettightedgeofpath@maybemoveto{\the\c@pgf@counta}{\the\c@pgf@countb}%

```

Second, create a line to the end point.

```

6367     \edef\forest@last@x{%
6368         \csname forest@pi@\forest@aplusone @the\c@pgf@countc x\endcsname}%
6369     \edef\forest@last@y{%
6370         \csname forest@pi@\forest@aplusone @the\c@pgf@countc y\endcsname}%
6371     \pgfsyssoftpath@lineto\forest@last@x\forest@last@y

```

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

```

6372     \c@pgf@countc=\csname forest@pi@\forest@aplusone @n\endcsname
6373     \c@pgf@countb=\csname forest@pi@the\c@pgf@counta @n\endcsname
6374 \fi
6375     \advance\c@pgf@countc 1
6376 \saferepeat
6377     \advance\c@pgf@countb 1
6378 }

```

\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.

```

6379 \def\forest@gettightedgeofpath@maybemoveto#1#2{%
6380     \forest@temptrue
6381     \ifx\forest@last@x\relax\else
6382         \ifdim\forest@last@x=\csname forest@pi@#1@#2x\endcsname\relax
6383         \ifdim\forest@last@y=\csname forest@pi@#1@#2y\endcsname\relax
6384             \forest@tempfalse
6385         \fi
6386     \fi
6387 \fi
6388 \ifforest@temp
6389     \edef\forest@last@x{\csname forest@pi@#1@#2x\endcsname}%
6390     \edef\forest@last@y{\csname forest@pi@#1@#2y\endcsname}%
6391     \pgfsyssoftpath@moveto\forest@last@x\forest@last@y
6392 \fi
6393 }

```

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

```

6394 \def\forest@simplifypath#1#2{%
6395     \pgfsyssoftpath@setcurrentpath\pgfutil@empty
6396     \forest@save\pgfsyssoftpath@tokendefs
6397     \let\pgfsyssoftpath@movetotoken\forest@simplifypath@moveto
6398     \let\pgfsyssoftpath@linetotoken\forest@simplifypath@lineto
6399     \let\forest@last@x\relax
6400     \let\forest@last@y\relax
6401     \let\forest@last@atan\relax
6402     #1%
6403     \ifx\forest@last@x\relax\else

```

```

6404 \ifx\forest@last@atan\relax\else
6405 \pgfsyssoftpath@lineto\forest@last@x\forest@last@y
6406 \fi
6407 \fi
6408 \forest@restore@pgfsyssoftpath@tokendefs
6409 \pgfsyssoftpath@getcurrentpath#2%
6410 \pgfsyssoftpath@setcurrentpath\pgfutil@empty
6411 }

```

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.

```

6412 \def\forest@simplifypath@moveto#1#2{%
6413 \ifx\forest@last@x\relax\else
6414 \pgfsyssoftpath@lineto\forest@last@x\forest@last@y
6415 \fi
6416 \pgfsyssoftpath@moveto{#1}{#2}%
6417 \def\forest@last@x{#1}%
6418 \def\forest@last@y{#2}%
6419 \let\forest@last@atan\relax
6420 }

```

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

```

6421 \def\forest@getedgeofpath@precision{1pt}

```

When a line-to is encountered...

```

6422 \def\forest@simplifypath@lineto#1#2{%
6423 \ifx\forest@last@x\relax

```

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

```

6424 \def\forest@last@x{#1}%
6425 \def\forest@last@y{#2}%
6426 \let\forest@last@atan\relax
6427 \else

```

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

```

6428 \pgfpointdiff{\pgfqpoint{#1}{#2}}{\pgfqpoint{\forest@last@x}{\forest@last@y}}%
6429 \ifdim\pgf@x<\pgfintersectiontolerance
6430 \ifdim-\pgf@x<\pgfintersectiontolerance
6431 \pgf@x=0pt
6432 \fi
6433 \fi
6434 \csname pgfmathatan2\endcsname{\pgf@x}{\pgf@y}%
6435 \let\forest@current@atan\pgfmathresult
6436 \ifx\forest@last@atan\relax

```

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

```

6437 \def\forest@last@x{#1}%
6438 \def\forest@last@y{#2}%
6439 \let\forest@last@atan\forest@current@atan
6440 \else

```

Otherwise, compare the first and the current slope.

```

6441 \pgfutil@tempdima=\forest@current@atan pt
6442 \advance\pgfutil@tempdima -\forest@last@atan pt
6443 \ifdim\pgfutil@tempdima<0pt\relax
6444 \multiply\pgfutil@tempdima -1
6445 \fi
6446 \ifdim\pgfutil@tempdima<\forest@getedgeofpath@precision\relax
6447 \else

```

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

```

6448      \pgfsyssoftpath@lineto\forest@last@x\forest@last@y
6449      \let\forest@last@atan\forest@current@atan
6450      \fi

```

In any event, update the last point.

```

6451      \def\forest@last@x{#1}%
6452      \def\forest@last@y{#2}%
6453      \fi
6454      \fi
6455  }

```

8.4 Get rectangle/band edge

```

6456 \def\forest@getnegativerectangleedgeofpath#1#2{%
6457   \forest@getnegativerectangleorbandedgeofpath{#1}{#2}{\the\pgf@xb}}
6458 \def\forest@getpositiverectangleedgeofpath#1#2{%
6459   \forest@getpositiverectangleorbandedgeofpath{#1}{#2}{\the\pgf@xb}}
6460 \def\forest@getbothrectangleedgesofpath#1#2#3{%
6461   \forest@getbothrectangleorbandedgesofpath{#1}{#2}{#3}{\the\pgf@xb}}
6462 \def\forest@bandlength{5000pt} % something large (ca. 180cm), but still manageable for TeX without producing
6463 \def\forest@getnegativebandedgeofpath#1#2{%
6464   \forest@getnegativerectangleorbandedgeofpath{#1}{#2}{\forest@bandlength}}
6465 \def\forest@getpositivebandedgeofpath#1#2{%
6466   \forest@getpositiverectangleorbandedgeofpath{#1}{#2}{\forest@bandlength}}
6467 \def\forest@getbothbandedgesofpath#1#2#3{%
6468   \forest@getbothrectangleorbandedgesofpath{#1}{#2}{#3}{\forest@bandlength}}
6469 \def\forest@getnegativerectangleorbandedgeofpath#1#2#3{%
6470   \forest@path@getboundingrectangle@ls#1{\forest@grow}%
6471   \edef\forest@gre@path{%
6472     \noexpand\pgfsyssoftpath@movetotoken{\the\pgf@xa}{\the\pgf@ya}%
6473     \noexpand\pgfsyssoftpath@linetotoken{#3}{\the\pgf@ya}%
6474   }%
6475   {%
6476     \pgftransformreset
6477     \pgftransformrotate{\forest@grow}%
6478     \forest@pgfpathtransformed\forest@gre@path
6479   }%
6480   \pgfsyssoftpath@getcurrentpath#2%
6481 }
6482 \def\forest@getpositiverectangleorbandedgeofpath#1#2#3{%
6483   \forest@path@getboundingrectangle@ls#1{\forest@grow}%
6484   \edef\forest@gre@path{%
6485     \noexpand\pgfsyssoftpath@movetotoken{\the\pgf@xa}{\the\pgf@yb}%
6486     \noexpand\pgfsyssoftpath@linetotoken{#3}{\the\pgf@yb}%
6487   }%
6488   {%
6489     \pgftransformreset
6490     \pgftransformrotate{\forest@grow}%
6491     \forest@pgfpathtransformed\forest@gre@path
6492   }%
6493   \pgfsyssoftpath@getcurrentpath#2%
6494 }
6495 \def\forest@getbothrectangleorbandedgesofpath#1#2#3#4{%
6496   \forest@path@getboundingrectangle@ls#1{\forest@grow}%
6497   \edef\forest@gre@negpath{%
6498     \noexpand\pgfsyssoftpath@movetotoken{\the\pgf@xa}{\the\pgf@ya}%
6499     \noexpand\pgfsyssoftpath@linetotoken{#4}{\the\pgf@ya}%
6500   }%
6501   \edef\forest@gre@pospath{%
6502     \noexpand\pgfsyssoftpath@movetotoken{\the\pgf@xa}{\the\pgf@yb}%
6503     \noexpand\pgfsyssoftpath@linetotoken{#4}{\the\pgf@yb}%

```

```

6504 }%
6505 {%
6506     \pgftransformreset
6507     \pgftransformrotate{\forest@grow}%
6508     \forest@pgfpathtransformed\forest@gre@negpath
6509 }%
6510 \pgfsyssoftpath@getcurrentpath#2%
6511 {%
6512     \pgftransformreset
6513     \pgftransformrotate{\forest@grow}%
6514     \forest@pgfpathtransformed\forest@gre@pospath
6515 }%
6516 \pgfsyssoftpath@getcurrentpath#3%
6517 }

```

8.5 Distance between paths

Another crucial part of the package.

```

6518 \def\forest@distance@between@edge@paths#1#2#3{%
6519     % #1, #2 = (edge) paths
6520     %
6521     % project paths
6522     \forest@projectpath@togrowline#1{\forest@p1@}%
6523     \forest@projectpath@togrowline#2{\forest@p2@}%
6524     % merge projections (the lists are sorted already, because edge
6525     % paths are |sorted|)
6526     \forest@dbep@mergeprojections
6527     {\forest@p1@}{\forest@p2@}%
6528     {\forest@P1@}{\forest@P2@}%
6529     % process projections
6530     \forest@processprojectioninfo{\forest@P1@}{\forest@PI1@}%
6531     \forest@processprojectioninfo{\forest@P2@}{\forest@PI2@}%
6532     % break paths
6533     \forest@breakpath#1{\forest@PI1@}\forest@broken@one
6534     \forest@breakpath#2{\forest@PI2@}\forest@broken@two
6535     % sort inner arrays ---optimize: it's enough to find max and min
6536     \forest@sort@inner@arrays{\forest@PI1@}\forest@sort@descending
6537     \forest@sort@inner@arrays{\forest@PI2@}\forest@sort@ascending
6538     % compute the distance
6539     \let\forest@distance\relax
6540     \c@pgf@countc=0
6541     \forest@loop
6542     \ifnum\c@pgf@countc<\csname forest@PI1@n\endcsname\relax
6543         \ifnum\csname forest@PI1@the\c@pgf@countc @n\endcsname=0 \else
6544             \ifnum\csname forest@PI2@the\c@pgf@countc @n\endcsname=0 \else
6545                 \pgfutil@tempdima=\csname forest@PI2@the\c@pgf@countc @0d\endcsname\relax
6546                 \advance\pgfutil@tempdima-\csname forest@PI1@the\c@pgf@countc @0d\endcsname\relax
6547                 \ifx\forest@distance\relax
6548                     \edef\forest@distance{\the\pgfutil@tempdima}%
6549                 \else
6550                     \ifdim\pgfutil@tempdima<\forest@distance\relax
6551                         \edef\forest@distance{\the\pgfutil@tempdima}%
6552                     \fi
6553                 \fi
6554             \fi
6555         \fi
6556         \advance\c@pgf@countc 1
6557     \forest@repeat
6558     \let#3\forest@distance
6559 }
6560 % merge projections: we need two projection arrays, both containing

```



```

6561 % projection points from both paths, but each with the original
6562 % points from only one path
6563 \def\forest@dbep@mergeprojections#1#2#3#4{%
6564 % TODO: optimize: v bistvu ni treba sortirat, ker je edge path e sortiran
6565 \forest@sortprojections{#1}%
6566 \forest@sortprojections{#2}%
6567 \c@pgf@counta=0
6568 \c@pgf@countb=0
6569 \c@pgf@countc=0
6570 \edef\forest@input@prefix@one{#1}%
6571 \edef\forest@input@prefix@two{#2}%
6572 \edef\forest@output@prefix@one{#3}%
6573 \edef\forest@output@prefix@two{#4}%
6574 \forest@dbep@mp@iterate
6575 \csedef{#3n}{\the\c@pgf@countc}%
6576 \csedef{#4n}{\the\c@pgf@countc}%
6577 }
6578 \def\forest@dbep@mp@iterate{%
6579 \let\forest@dbep@mp@next\forest@dbep@mp@iterate
6580 \ifnum\c@pgf@counta<\csname\forest@input@prefix@one n\endcsname\relax
6581 \ifnum\c@pgf@countb<\csname\forest@input@prefix@two n\endcsname\relax
6582 \let\forest@dbep@mp@next\forest@dbep@mp@do
6583 \else
6584 \let\forest@dbep@mp@next\forest@dbep@mp@iteratefirst
6585 \fi
6586 \else
6587 \ifnum\c@pgf@countb<\csname\forest@input@prefix@two n\endcsname\relax
6588 \let\forest@dbep@mp@next\forest@dbep@mp@iteratesecond
6589 \else
6590 \let\forest@dbep@mp@next\relax
6591 \fi
6592 \fi
6593 \forest@dbep@mp@next
6594 }
6595 \def\forest@dbep@mp@do{%
6596 \forest@sort@cmptwodimcs%
6597 {\forest@input@prefix@one\the\c@pgf@counta xp}%
6598 {\forest@input@prefix@one\the\c@pgf@counta yp}%
6599 {\forest@input@prefix@two\the\c@pgf@countb xp}%
6600 {\forest@input@prefix@two\the\c@pgf@countb yp}%
6601 \if\forest@sort@cmp@result=%
6602 \forest@dbep@mp@store@p\forest@input@prefix@one\c@pgf@counta
6603 \forest@dbep@mp@store@o\forest@input@prefix@one
6604 \c@pgf@counta\forest@output@prefix@one
6605 \forest@dbep@mp@store@o\forest@input@prefix@two
6606 \c@pgf@countb\forest@output@prefix@two
6607 \advance\c@pgf@counta 1
6608 \advance\c@pgf@countb 1
6609 \else
6610 \if\forest@sort@cmp@result>%
6611 \forest@dbep@mp@store@p\forest@input@prefix@two\c@pgf@countb
6612 \forest@dbep@mp@store@o\forest@input@prefix@two
6613 \c@pgf@countb\forest@output@prefix@two
6614 \advance\c@pgf@countb 1
6615 \else%<
6616 \forest@dbep@mp@store@p\forest@input@prefix@one\c@pgf@counta
6617 \forest@dbep@mp@store@o\forest@input@prefix@one
6618 \c@pgf@counta\forest@output@prefix@one
6619 \advance\c@pgf@counta 1
6620 \fi
6621 \fi

```

```

6622 \advance\c@pgf@countc 1
6623 \forest@dbep@mp@iterate
6624 }
6625 \def\forest@dbep@mp@@store@p#1#2{%
6626 \csletcs
6627 {\forest@output@prefix@one\the\c@pgf@countc xp}%
6628 {#1\the#2xp}%
6629 \csletcs
6630 {\forest@output@prefix@one\the\c@pgf@countc yp}%
6631 {#1\the#2yp}%
6632 \csletcs
6633 {\forest@output@prefix@two\the\c@pgf@countc xp}%
6634 {#1\the#2xp}%
6635 \csletcs
6636 {\forest@output@prefix@two\the\c@pgf@countc yp}%
6637 {#1\the#2yp}%
6638 }
6639 \def\forest@dbep@mp@@store@o#1#2#3{%
6640 \csletcs{#3\the\c@pgf@countc xo}{#1\the#2xo}%
6641 \csletcs{#3\the\c@pgf@countc yo}{#1\the#2yo}%
6642 }
6643 \def\forest@dbep@mp@iteratefirst{%
6644 \forest@dbep@mp@iterateone\forest@input@prefix@one\c@pgf@counta\forest@output@prefix@one
6645 }
6646 \def\forest@dbep@mp@iteratesecond{%
6647 \forest@dbep@mp@iterateone\forest@input@prefix@two\c@pgf@countb\forest@output@prefix@two
6648 }
6649 \def\forest@dbep@mp@iterateone#1#2#3{%
6650 \forest@loop
6651 \ifnum#2<\csname#1n\endcsname\relax
6652 \forest@dbep@mp@@store@p#1#2%
6653 \forest@dbep@mp@@store@o#1#2#3%
6654 \advance\c@pgf@countc 1
6655 \advance#21
6656 \forest@repeat
6657 }

```

8.6 Utilities

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

```

6658 \newif\ifforest@equaltotolerance
6659 \def\forest@equaltotolerance#1#2{%
6660 \pgfpointdiff{#1}{#2}%
6661 \ifdim\pgf@x<0pt \multiply\pgf@x -1 \fi
6662 \ifdim\pgf@y<0pt \multiply\pgf@y -1 \fi
6663 \global\forest@equaltolerancefalse
6664 \ifdim\pgf@x<\pgfintersectiontolerance\relax
6665 \ifdim\pgf@y<\pgfintersectiontolerance\relax
6666 \global\forest@equaltolerancetrue
6667 \fi
6668 \fi
6669 }}

```

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

```

6670 \def\forest@save@pgfsyssoftpath@tokendefs{%
6671 \let\forest@origmovetotoken\pgfsyssoftpath@movetotoken
6672 \let\forest@origlinetotoken\pgfsyssoftpath@linetotoken
6673 \let\forest@origcurvetosupportatoken\pgfsyssoftpath@curvetosupportatoken
6674 \let\forest@origcurvetosupportbtoken\pgfsyssoftpath@curvetosupportbtoken
6675 \let\forest@origcurvetotoken\pgfsyssoftpath@curvetototoken

```

```

6676 \let\forest@origrectcornertoken\pgfsyssoftpath@rectcornertoken
6677 \let\forest@origrectsizetoken\pgfsyssoftpath@rectsizetoken
6678 \let\forest@origclosepathtoken\pgfsyssoftpath@closepathtoken
6679 \let\pgfsyssoftpath@movetotoken\forest@badtoken
6680 \let\pgfsyssoftpath@linetotoken\forest@badtoken
6681 \let\pgfsyssoftpath@curvetosupportatoken\forest@badtoken
6682 \let\pgfsyssoftpath@curvetosupportbtoken\forest@badtoken
6683 \let\pgfsyssoftpath@curvetototoken\forest@badtoken
6684 \let\pgfsyssoftpath@rectcornertoken\forest@badtoken
6685 \let\pgfsyssoftpath@rectsizetoken\forest@badtoken
6686 \let\pgfsyssoftpath@closepathtoken\forest@badtoken
6687 }
6688 \def\forest@badtoken{%
6689 \PackageError{forest}{This token should not be in this path}{}%
6690 }
6691 \def\forest@restore@pgfsyssoftpath@tokendefs{%
6692 \let\pgfsyssoftpath@movetotoken\forest@origmovetotoken
6693 \let\pgfsyssoftpath@linetotoken\forest@origlinetotoken
6694 \let\pgfsyssoftpath@curvetosupportatoken\forest@origcurvetosupportatoken
6695 \let\pgfsyssoftpath@curvetosupportbtoken\forest@origcurvetosupportbtoken
6696 \let\pgfsyssoftpath@curvetototoken\forest@origcurvetotoken
6697 \let\pgfsyssoftpath@rectcornertoken\forest@origrectcornertoken
6698 \let\pgfsyssoftpath@rectsizetoken\forest@origrectsizetoken
6699 \let\pgfsyssoftpath@closepathtoken\forest@origclosepathtoken
6700 }

```

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

```

6701 \def\forest@extendpath#1#2#3{%
6702 \pgf@process{#3}%
6703 \pgfsyssoftpath@setcurrentpath#1%
6704 \forest@save@pgfsyssoftpath@tokendefs
6705 \let\pgfsyssoftpath@movetotoken\forest@extendpath@moveto
6706 \let\pgfsyssoftpath@linetotoken\forest@extendpath@lineto
6707 #2%
6708 \forest@restore@pgfsyssoftpath@tokendefs
6709 \pgfsyssoftpath@getcurrentpath#1%
6710 }
6711 \def\forest@extendpath@moveto#1#2{%
6712 \forest@extendpath@do{#1}{#2}\pgfsyssoftpath@moveto
6713 }
6714 \def\forest@extendpath@lineto#1#2{%
6715 \forest@extendpath@do{#1}{#2}\pgfsyssoftpath@lineto
6716 }
6717 \def\forest@extendpath@do#1#2#3{%
6718 {%
6719 \advance\pgf@x #1
6720 \advance\pgf@y #2
6721 #3{\the\pgf@x}{\the\pgf@y}%
6722 }%
6723 }

```

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.)

```

6724 \def\forest@path@getboundingrectangle@ls#1#2{%
6725 {%
6726 \pgftransformreset
6727 \pgftransformrotate{-(#2)}%
6728 \forest@pgfpathtransformed#1%
6729 }%
6730 \pgfsyssoftpath@getcurrentpath\forest@gbr@rotatedpath
6731 \forest@path@getboundingrectangle@xy\forest@gbr@rotatedpath

```

```

6732 }
6733 \def\forest@path@getboundingrectangle@xy#1{%
6734   \forest@save@pgfsyssoftpath@tokendefs
6735   \let\pgfsyssoftpath@movetotoken\forest@gbr@firstpoint
6736   \let\pgfsyssoftpath@linetotoken\forest@gbr@firstpoint
6737   #1%
6738   \forest@restore@pgfsyssoftpath@tokendefs
6739 }
6740 \def\forest@gbr@firstpoint#1#2{%
6741   \pgf@xa=#1 \pgf@xb=#1 \pgf@ya=#2 \pgf@yb=#2
6742   \let\pgfsyssoftpath@movetotoken\forest@gbr@point
6743   \let\pgfsyssoftpath@linetotoken\forest@gbr@point
6744 }
6745 \def\forest@gbr@point#1#2{%
6746   \ifdim#1<\pgf@xa\relax\pgf@xa=#1 \fi
6747   \ifdim#1>\pgf@xb\relax\pgf@xb=#1 \fi
6748   \ifdim#2<\pgf@ya\relax\pgf@ya=#2 \fi
6749   \ifdim#2>\pgf@yb\relax\pgf@yb=#2 \fi
6750 }

```

9 The outer UI

9.1 Externalization

```

6751 \pgfkeys{/forest/external/.cd,
6752   %copy command/.initial={cp "\source" "\target"},
6753   copy command/.initial={},
6754   optimize/.is if=forest@external@optimize@,
6755   context/.initial={%
6756     \forestOve{\csname forest@id@of@standard node\endcsname}{environment@formula}},
6757   depends on macro/.style={context/.append/.expanded={%
6758     \expandafter\detokenize\expandafter{#1}}},
6759 }
6760 \def\forest@file@copy#1#2{%
6761   \IfFileExists{#1}{%
6762     \pgfkeysgetvalue{/forest/external/copy command}\forest@copy@command
6763     \ifdefempty\forest@copy@command{%
6764       \forest@file@copy@{#1}{#2}%
6765     }{ % copy by external command
6766       \def\source{#1}%
6767       \def\target{#2}%
6768       \immediate\write18{\forest@copy@command}%
6769     }%
6770   }{}%
6771 }
6772 \def\forest@file@copy@#1#2{%
6773   \newread\forest@copy@in
6774   \openin\forest@copy@in=#1
6775   \newwrite\forest@copy@out
6776   \immediate\openout\forest@copy@out=#2
6777   \endlinechar-1
6778   \loop
6779   \unless\ifeof\forest@copy@in
6780     \readline\forest@copy@in to\forest@temp
6781     \immediate\write\forest@copy@out{\forest@temp}%
6782   \repeat
6783   \immediate\closeout\forest@copy@out
6784   \closein\forest@copy@in
6785 }
6786 \newif\ifforest@external@optimize@

```

```

6787 \forest@external@optimize@true
6788 \ifforest@install@keys@to@tikz@path@
6789 \tikzset{
6790   fit to/.style={
6791     /forest/for nodewalk=%
6792     {TeX={\def\forest@fitto{}}\#1}%
6793     {TeX={\eappto\forest@fitto{(\forestove{name})}}},
6794     fit/.expanded={\forest@fitto}
6795   },
6796 }
6797 \fi
6798 \ifforest@external@
6799 \ifdefined\tikzexternal@tikz@replacement\else
6800   \usetikzlibrary{external}%
6801 \fi
6802 \pgfkeys{%
6803   /tikz/external/failed ref warnings for={},
6804   /pgf/images/aux in dpth=false,
6805 }%
6806 \tikzifexternalizing{}{%
6807   \forest@file@copy{\jobname.aux}{\jobname.aux.copy}%
6808 }%
6809 \AtBeginDocument{%
6810   \tikzifexternalizing{%
6811     \IfFileExists{\tikzexternalrealjob.aux.copy}{%
6812       \makeatletter
6813       \input\tikzexternalrealjob.aux.copy\relax
6814       \makeatother
6815     }{}%
6816   }{%
6817     \newwrite\forest@auxout
6818     \immediate\openout\forest@auxout=\tikzexternalrealjob.for.tmp
6819   }%
6820   \IfFileExists{\tikzexternalrealjob.for}{%
6821     {%
6822       \makehashother\makeatletter
6823       \input\tikzexternalrealjob.for\relax
6824     }%
6825   }{}%
6826 }%
6827 \AtEndDocument{%
6828   \tikzifexternalizing{}{%
6829     \immediate\closeout\forest@auxout
6830     \forest@file@copy{\jobname.for.tmp}{\jobname.for}%
6831   }%
6832 }%
6833 \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.

```

6834 \NewDocumentEnvironment{forest}{D(){} }{%
6835   \forest@config{#1}%
6836   \Collect@Body
6837   \forest@env
6838 }{}
6839 \NewDocumentCommand{\Forest}{s D(){} m}{%
6840   \forest@config{#2}%
6841   \IfBooleanTF{#1}{\let\forest@next\forest@env}{\let\forest@next\forest@group@env}%

```

```

6842 \forest@next{#3}%
6843 }
6844 \def\forest@config#1{%
6845 \def\forest@stages{stages}%
6846 \forestset{@config/.cd,#1}%
6847 }
6848 \forestset{@config/.cd,
6849 stages/.store in=\forest@stages,
6850 .unknown/.code={\PackageError{forest}{Unknown config option for forest environment/command.}{In Forest v2.0
6851 }
6852 \def\forest@group@env#1{{\forest@env{#1}}}
6853 \newif\ifforest@externalize@tree@
6854 \newif\ifforest@was@tikzexternalwasenable
6855 \newcommand\forest@env[1]{%
6856 \let\forest@external@next\forest@begin
6857 \forest@was@tikzexternalwasenablefalse
6858 \ifdefined\tikzexternal@tikz@replacement
6859 \ifx\tikz\tikzexternal@tikz@replacement
6860 \forest@was@tikzexternalwasenabletrue
6861 \tikzexternaldisable
6862 \fi
6863 \fi
6864 \forest@externalize@tree@false
6865 \ifforest@external@
6866 \ifforest@was@tikzexternalwasenable
6867 \forest@env@
6868 \fi
6869 \fi
6870 \forest@standardnode@calibrate
6871 \forest@external@next{#1}%
6872 }
6873 \def\forest@env@{%
6874 \iftikzexternalexportnext
6875 \tikzifexternalizing{%
6876 \let\forest@external@next\forest@begin@externalizing
6877 }{%
6878 \let\forest@external@next\forest@begin@externalize
6879 }%
6880 \else
6881 \tikzexternalexportnexttrue
6882 \fi
6883 }

```

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

```

6884 \long\def\forest@begin@externalizing#1{%
6885 \forest@external@setup{#1}%
6886 \let\forest@external@next\forest@begin
6887 \forest@externalize@inner@n=-1
6888 \ifforest@external@optimize@\forest@externalizing@maybeoptimize\fi
6889 \forest@external@next{#1}%
6890 \tikzexternalenable
6891 }
6892 \def\forest@externalizing@maybeoptimize{%
6893 \edef\forest@temp{\tikzexternalrealjob-forest-\forest@externalize@outer@n}%
6894 \edef\forest@marshal{%
6895 \noexpand\pgfutil@in@
6896 {\expandafter\detokenize\expandafter{\forest@temp}.}
6897 {\expandafter\detokenize\expandafter{\pgfactualjobname}.}%
6898 }\forest@marshal
6899 \ifpgfutil@in@
6900 \else

```

```

6901 \let\forest@external@next\@gobble
6902 \fi
6903 }

```

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

```

6904 \long\def\forest@begin@externalize#1{%
6905 \forest@external@setup{#1}%
6906 \iftikzexternal@file@isuptodate
6907 \setbox0=\hbox{%
6908 \csname forest@externalcheck@\forest@externalize@outer@n\endcsname
6909 }%
6910 \fi
6911 \iftikzexternal@file@isuptodate
6912 \csname forest@externalload@\forest@externalize@outer@n\endcsname
6913 \else
6914 \forest@externalize@tree@true
6915 \forest@externalize@inner@n=-1
6916 \forest@begin{#1}%
6917 \ifcsdef{forest@externalize@@\forest@externalize@id}{}%
6918 \immediate\write\forest@auxout{%
6919 \noexpand\forest@external
6920 {\forest@externalize@outer@n}%
6921 {\expandafter\detokenize\expandafter{\forest@externalize@id}}%
6922 {\expandonce\forest@externalize@checkimages}%
6923 {\expandonce\forest@externalize@loadimages}%
6924 }%
6925 }%
6926 \fi
6927 \tikzexternalenable
6928 }
6929 \def\forest@includeexternal@check#1{%
6930 \tikzsetnextfilename{#1}%
6931 \IfFileExists{\tikzexternal@filenameprefix/#1}{\tikzexternal@file@isuptodatetrue}{\tikzexternal@file@isuptodatefalse}
6932 }
6933 \def\makehashother{\catcode'\#12}%
6934 \long\def\forest@external@setup#1{%
6935 % set up \forest@externalize@id and \forest@externalize@outer@n
6936 % we need to deal with #s correctly (\write doubles them)
6937 \setbox0=\hbox{\makehashother\makeatletter
6938 \scantokens{\forest@temp@toks{#1}}\expandafter
6939 }%
6940 \expandafter\forest@temp@toks\expandafter{\the\forest@temp@toks}%
6941 \edef\forest@temp{\pgfkeysvalueof{/forest/external/context}}%
6942 \edef\forest@externalize@id{%
6943 \expandafter\detokenize\expandafter{\forest@temp}%
6944 @@%
6945 \expandafter\detokenize\expandafter{\the\forest@temp@toks}%
6946 }%
6947 \letcs\forest@externalize@outer@n\forest@externalize@@\forest@externalize@id%
6948 \ifdefined\forest@externalize@outer@n
6949 \global\tikzexternal@file@isuptodatetrue
6950 \else
6951 \global\advance\forest@externalize@max@outer@n 1
6952 \edef\forest@externalize@outer@n{\the\forest@externalize@max@outer@n}%
6953 \global\tikzexternal@file@isuptodatefalse
6954 \fi
6955 \def\forest@externalize@loadimages{}%
6956 \def\forest@externalize@checkimages{}%
6957 }
6958 \newcount\forest@externalize@max@outer@n
6959 \global\forest@externalize@max@outer@n=0

```

6960 \newcount\forest@externalize@inner@n

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

```

6961 \long\def\forest@external#1#2#3#4{% #1=n,#2=context+source code,#3=update check code, #4=load code
6962   \ifnum\forest@externalize@max@outer@n<#1
6963     \global\forest@externalize@max@outer@n=#1
6964   \fi
6965   \global\csdef{forest@externalize@@\detokenize{#2}}{#1}%
6966   \global\csdef{forest@externalcheck@#1}{#3}%
6967   \global\csdef{forest@externalload@#1}{#4}%
6968   \tikzifexternalizing{}{%
6969     \immediate\write\forest@auxout{%
6970       \noexpand\forest@external{#1}%
6971       {\expandafter\detokenize\expandafter{#2}}}%
6972     {\unexpanded{#3}}}%
6973     {\unexpanded{#4}}}%
6974   }%
6975 }%
6976 }

```

These two macros include the external picture.

```

6977 \def\forest@includeexternal#1{%
6978   \edef\forest@temp{\pgfkeysvalueof{/forest/external/context}}%
6979   %\typeout{forest: Including external picture '#1' for forest context+code: '\expandafter\detokenize\expand
6980   {%
6981     %\def\pgf@declaredraftimage##1##2{\def\pgf@image{\hbox{}}}%
6982     \tikzsetnextfilename{#1}%
6983     \tikzexternalenable
6984     \tikz{}%
6985   }%
6986 }
6987 \def\forest@includeexternal@box#1#2{%
6988   \global\setbox#1=\hbox{\forest@includeexternal{#2}}%
6989 }

```

This code runs the bracket parser and stage processing.

```

6990 \long\def\forest@begin#1{%
6991   \iffalse{\fi\forest@parsebracket#1}%
6992 }
6993 \def\forest@parsebracket{%
6994   \bracketParse{\forest@get@root@afterthought}\forest@root=%
6995 }
6996 \def\forest@get@root@afterthought{%
6997   \expandafter\forest@get@root@afterthought@\expandafter{\iffalse}\fi
6998 }
6999 \long\def\forest@get@root@afterthought@#1{%
7000   \ifblank{#1}{\fi}%
7001   \forest@eappto{\forest@root}{given options}{,afterthought={\unexpanded{#1}}}%
7002 }%
7003 \forest@do
7004 }
7005 \def\forest@do{%
7006   \forest@node@Compute@numeric@ts@info{\forest@root}%
7007   \expandafter\forestset\expandafter{\forest@stages}%
7008   \ifforest@was@tikzexternalwasenable
7009     \tikzexternalenable
7010   \fi
7011 }

```


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!

```

7012 \def\forest@standardnode@new{%
7013   \advance\forest@node@maxid1
7014   \forest@fornode{\the\forest@node@maxid}{%
7015     \forest@node@init
7016     \forestoeset{id}{\forest@cn}%
7017     \forest@node@setname@silent{standard node}%
7018   }%
7019 }
7020 \def\forest@standardnode@calibrate{%
7021   \forest@fornode{\forest@node@Nametoid{standard node}}{%
7022     \edef\forest@environment{\forestove{environment@formula}}%
7023     \forestoget{previous@environment}\forest@previous@environment
7024     \ifx\forest@environment\forest@previous@environment\else
7025       \forestolet{previous@environment}\forest@environment
7026       \forest@node@typeset
7027       \forestoget{calibration@procedure}\forest@temp
7028       \expandafter\forestset\expandafter{\forest@temp}%
7029     \fi
7030   }%
7031 }

```

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.)

```

7032 \def\forestStandardNode[#1]#2#3#4{%
7033   \let\forest@standardnode@restoretikzexternal\relax
7034   \ifdefined\tikzexternaldisable
7035     \ifx\tikz\tikzexternal\tikz@replacement
7036       \tikzexternaldisable
7037       \let\forest@standardnode@restoretikzexternal\tikzexternalenable
7038     \fi
7039   \fi
7040   \forest@standardnode@new
7041   \forest@fornode{\forest@node@Nametoid{standard node}}{%
7042     \forestset{content=#1}%
7043     \forestoset{environment@formula}{#2}%
7044     \edef\forest@temp{\unexpanded{#3}}%
7045     \forestolet{calibration@procedure}\forest@temp
7046     \def\forest@calibration@initializing@code{%
7047       \pgfqkeys{/forest/initializing@code}{#4}%
7048       \forestolet{initializing@code}\forest@calibration@initializing@code
7049     \forest@standardnode@restoretikzexternal
7050   }
7051 }
7052 \forestset{initializing@code/.unknown/.code={%
7053   \eappto\forest@calibration@initializing@code{%
7054     \noexpand\forestoget{\forest@node@Nametoid{standard node}}{\pgfkeyscurrentname}\noexpand\forest@temp
7055     \noexpand\forestolet{\pgfkeyscurrentname}\noexpand\forest@temp
7056   }%
7057 }
7058 }

```

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

```

7059 \def\forest@initializefromstandardnode{%
7060   \forestOve{\forest@node@Nametoid{standard node}}{initializing@code}%
7061 }

```

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`.)

```

7062 \forestStandardNode[dj]
7063 {%
7064   \forestOve{\forest@node@Nametoid{standard node}}{content},%
7065   \the\ht\strutbox,\the\pgflinewidth,%
7066   \pgfkeysvalueof{/pgf/inner ysep},\pgfkeysvalueof{/pgf/outer ysep},%
7067   \pgfkeysvalueof{/pgf/inner xsep},\pgfkeysvalueof{/pgf/outer xsep}%
7068 }
7069 {
7070   l sep={\the\ht\strutbox+\pgfkeysvalueof{/pgf/inner ysep}},
7071   l={l sep()+abs(max_y()-min_y())+2*\pgfkeysvalueof{/pgf/outer ysep}},
7072   s sep={2*\pgfkeysvalueof{/pgf/inner xsep}}
7073 }
7074 {l sep,l,s sep}

```

9.4 ls coordinate system

```

7075 \pgfqkeys{/forest/@cs}{%
7076   name/.code={%
7077     \edef\forest@cn{\forest@node@Nametoid{#1}}%
7078     \forest@forestcs@resetxy},
7079   id/.code={%
7080     \edef\forest@cn{#1}%
7081     \forest@forestcs@resetxy},
7082   go/.code={%
7083     \forest@go{#1}%
7084     \forest@forestcs@resetxy},
7085   anchor/.code={\forest@forestcs@anchor{#1}},
7086   l/.code={%
7087     \pgfmathsetlengthmacro\forest@forestcs@l{#1}%
7088     \forest@forestcs@ls
7089   },
7090   s/.code={%
7091     \pgfmathsetlengthmacro\forest@forestcs@s{#1}%
7092     \forest@forestcs@ls
7093   },
7094   .unknown/.code={%
7095     \expandafter\pgfutil@in@\expandafter.\expandafter{\pgfkeyscurrentname}%
7096     \ifpgfutil@in@
7097       \expandafter\forest@forestcs@namegoanchor\pgfkeyscurrentname\forest@end
7098     \else
7099       \expandafter\forest@nameandgo\expandafter{\pgfkeyscurrentname}%
7100       \forest@forestcs@resetxy
7101     \fi
7102   }
7103 }
7104 \def\forest@forestcs@resetxy{%
7105   \ifnum\forest@cn=0 \forest@cs@invalidnodeerror\fi

```

```

7106 \global\pgf@x\forestove{x}%
7107 \global\pgf@y\forestove{y}%
7108 }
7109 \def\forest@forestcs@ls{%
7110 \ifdefined\forest@forestcs@l
7111 \ifdefined\forest@forestcs@s
7112 {%
7113 \pgftransformreset
7114 \pgftransformrotate{\forestove{grow}}%
7115 \pgfpointransformed{\pgfpoint{\forest@forestcs@l}{\forest@forestcs@s}}%
7116 }%
7117 \global\advance\pgf@x\forestove{x}%
7118 \global\advance\pgf@y\forestove{y}%
7119 \fi
7120 \fi
7121 }
7122 \def\forest@forestcs@anchor#1{%
7123 \edef\forest@marshal{%
7124 \noexpand\forest@original@tikz@parse@node\relax
7125 (\forestove{name}\ifx\relax#1\relax\else.\fi#1)%
7126 }\forest@marshal
7127 }
7128 \def\forest@forestcs@namegoanchor#1.#2\forest@end{%
7129 \forest@nameandgo{#1}%
7130 \ifnum\forest@cn=0 \forest@cs@invalidnodeerror\fi
7131 \forest@forestcs@anchor{#2}%
7132 }
7133 \def\forest@cs@invalidnodeerror{%
7134 \PackageError{forest}{Attempt to refer to the invalid node by "forest cs"}{}%
7135 }
7136 \tikzdeclarecoordinatesystem{forest}{%
7137 \forest@forthis{%
7138 \forest@forestcs@resetxy
7139 \ifdefined\forest@forestcs@l\undef\forest@forestcs@l\fi
7140 \ifdefined\forest@forestcs@s\undef\forest@forestcs@s\fi
7141 \pgfqkeys{/forest/@cs}{#1}%
7142 }%
7143 }

```

9.5 Relative node names in TikZ

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

```

7144 \def\forest@tikz@parse@node#1(#2){%
7145 \pgfutil@in@.#2}%
7146 \ifpgfutil@in@
7147 \expandafter\forest@tikz@parse@node@checkiftikzname@withdot
7148 \else%
7149 \expandafter\forest@tikz@parse@node@checkiftikzname@withoutdot
7150 \fi%
7151 #1(#2)\forest@end
7152 }
7153 \def\forest@tikz@parse@node@checkiftikzname@withdot#1(#2.#3)\forest@end{%
7154 \forest@tikz@parse@node@checkiftikzname#1{#2}{.#3}}
7155 \def\forest@tikz@parse@node@checkiftikzname@withoutdot#1(#2)\forest@end{%
7156 \forest@tikz@parse@node@checkiftikzname#1{#2}{}}
7157 \def\forest@tikz@parse@node@checkiftikzname#1#2#3{%
7158 \expandafter\ifx\csname pgf@sh@ns@#2\endcsname\relax
7159 \forest@forthis{%
7160 \forest@nameandgo{#2}%
7161 \ifnum\forest@cn=0 \forest@cs@invalidnodeerror\fi
7162 \edef\forest@temp@relativenodename{\forestove{name}}%

```

```

7163 }%
7164 \else
7165 \def\forest@temp@relativenodename{#2}%
7166 \fi
7167 \expandafter\forest@original@tikz@parse@node\expandafter#1\expandafter(\forest@temp@relativenodename#3)%
7168 }
7169 \def\forest@nameandgo#1{%
7170 \pgfutil@in@{#1}%
7171 \ifpgfutil@in@
7172 \forest@nameandgo@(#1)%
7173 \else
7174 \ifstrempy{#1}{-}{\edef\forest@cn{\forest@node@Nametoid{#1}}}%
7175 \fi
7176 }
7177 \def\forest@nameandgo@(#1!#2){%
7178 \ifstrempy{#1}{-}{\edef\forest@cn{\forest@node@Nametoid{#1}}}%
7179 \forest@go{#2}%
7180 }

```

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.

```

7181 \forestset{
7182   declare count={rotate}{0},
7183   autoforward'={rotate}{node options},
7184 }

```

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

```

7185 \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.

```

7186 \newif\ifforest@anchor@forwardtotikz

```

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

```

7187 \newif\ifforest@anchor@isborder

```

The UI macro.

```

7188 \def\forestanchortotikzanchor#1#2{% #1 = forest anchor, #2 = macro to receive the tikz anchor
7189 \forest@anchor@forwardtotikzfalse
7190 \forest@anchor@do{#1}{\forest@cn}%
7191 \let#2\forest@temp@anchor
7192 }

```

Generic anchors.

```

7193 \pgfdeclaregenericanchor{child anchor}{%
7194 \forest@anchor@forwardtotikztrue
7195 \forest@anchor@do{#1}{child anchor}{\forest@referencednodeid}%
7196 }
7197 \pgfdeclaregenericanchor{parent anchor}{%
7198 \forest@anchor@forwardtotikztrue
7199 \forest@anchor@do{#1}{parent anchor}{\forest@referencednodeid}%
7200 }
7201 \pgfdeclaregenericanchor{anchor}{%
7202 \forest@anchor@forwardtotikztrue
7203 \forest@anchor@do{#1}{anchor}{\forest@referencednodeid}%
7204 }
7205 \pgfdeclaregenericanchor{children}{%

```

```

7206 \forest@anchor@forwardtotikztrue
7207 \forest@anchor@do{#1}{children}{\forest@referencednodeid}%
7208 }
7209 \pgfdeclaregenericanchor{children first}{%
7210 \forest@anchor@forwardtotikztrue
7211 \forest@anchor@do{#1}{children first}{\forest@referencednodeid}%
7212 }
7213 \pgfdeclaregenericanchor{first}{%
7214 \forest@anchor@forwardtotikztrue
7215 \forest@anchor@do{#1}{first}{\forest@referencednodeid}%
7216 }
7217 \pgfdeclaregenericanchor{parent first}{%
7218 \forest@anchor@forwardtotikztrue
7219 \forest@anchor@do{#1}{parent first}{\forest@referencednodeid}%
7220 }
7221 \pgfdeclaregenericanchor{parent}{%
7222 \forest@anchor@forwardtotikztrue
7223 \forest@anchor@do{#1}{parent}{\forest@referencednodeid}%
7224 }
7225 \pgfdeclaregenericanchor{parent last}{%
7226 \forest@anchor@forwardtotikztrue
7227 \forest@anchor@do{#1}{parent last}{\forest@referencednodeid}%
7228 }
7229 \pgfdeclaregenericanchor{last}{%
7230 \forest@anchor@forwardtotikztrue
7231 \forest@anchor@do{#1}{last}{\forest@referencednodeid}%
7232 }
7233 \pgfdeclaregenericanchor{children last}{%
7234 \forest@anchor@forwardtotikztrue
7235 \forest@anchor@do{#1}{children last}{\forest@referencednodeid}%
7236 }
7237 \pgfdeclaregenericanchor{children'}{%
7238 \forest@anchor@forwardtotikztrue
7239 \forest@anchor@do{#1}{children'}{\forest@referencednodeid}%
7240 }
7241 \pgfdeclaregenericanchor{children first'}{%
7242 \forest@anchor@forwardtotikztrue
7243 \forest@anchor@do{#1}{children first'}{\forest@referencednodeid}%
7244 }
7245 \pgfdeclaregenericanchor{first'}{%
7246 \forest@anchor@forwardtotikztrue
7247 \forest@anchor@do{#1}{first'}{\forest@referencednodeid}%
7248 }
7249 \pgfdeclaregenericanchor{parent first'}{%
7250 \forest@anchor@forwardtotikztrue
7251 \forest@anchor@do{#1}{parent first'}{\forest@referencednodeid}%
7252 }
7253 \pgfdeclaregenericanchor{parent'}{%
7254 \forest@anchor@forwardtotikztrue
7255 \forest@anchor@do{#1}{parent'}{\forest@referencednodeid}%
7256 }
7257 \pgfdeclaregenericanchor{parent last'}{%
7258 \forest@anchor@forwardtotikztrue
7259 \forest@anchor@do{#1}{parent last'}{\forest@referencednodeid}%
7260 }
7261 \pgfdeclaregenericanchor{last'}{%
7262 \forest@anchor@forwardtotikztrue
7263 \forest@anchor@do{#1}{last'}{\forest@referencednodeid}%
7264 }
7265 \pgfdeclaregenericanchor{children last'}{%
7266 \forest@anchor@forwardtotikztrue

```

```

7267 \forest@anchor@do{#1}{children last'}{\forest@referencednodeid}%
7268 }

```

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

```

7269 \def\forest@anchor@do#1#2#3{% #1 = shape name, #2 = (potentially) forest anchor, #3 = node id
7270 \forest@for node{#3}{%
7271 \def\forest@temp@anchor{#2}%
7272 \forest@anchor@snapbordertocompassfalse
7273 \forest@anchor@isborderfalse
7274 \forest@anchor@to@tikz@anchor
7275 \forest@anchor@border@to@compass
7276 \ifforest@anchor@forwardtotikz
7277 \forest@anchor@forward{#1}%
7278 \else
7279 \fi
7280 }%
7281 }

```

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

```

7282 \def\forest@anchor@to@tikz@anchor{%
7283 \ifcsdef{forest@anchor@@\forest@temp@anchor}{%
7284 \csuse{forest@anchor@@\forest@temp@anchor}%
7285 \forest@anchor@to@tikz@anchor
7286 }{}%
7287 }

```

Actual computation.

```

7288 \csdef{forest@anchor@@parent anchor}{%
7289 \forest@toget{parent anchor}\forest@temp@anchor}
7290 \csdef{forest@anchor@@child anchor}{%
7291 \forest@toget{child anchor}\forest@temp@anchor}
7292 \csdef{forest@anchor@@anchor}{%
7293 \forest@toget{anchor}\forest@temp@anchor}
7294 \csdef{forest@anchor@@children'}{%
7295 \forest@anchor@isbordertrue
7296 \edef\forest@temp@anchor{\number\numexpr\forest@ve{grow}-\forest@ve{rotate}}%
7297 }
7298 \csdef{forest@anchor@@parent'}{%
7299 \forest@anchor@isbordertrue
7300 \edef\forest@temp@anchor{\number\numexpr\forest@ve{grow}-\forest@ve{rotate}+180}%
7301 }
7302 \csdef{forest@anchor@@first'}{%
7303 \forest@anchor@isbordertrue
7304 \edef\forest@temp@anchor{\number\numexpr\forest@ve{grow}-\forest@ve{rotate}\ifnum\forest@ve{reversed}=0 -\e
7305 }
7306 \csdef{forest@anchor@@last'}{%
7307 \forest@anchor@isbordertrue
7308 \edef\forest@temp@anchor{\number\numexpr\forest@ve{grow}-\forest@ve{rotate}\ifnum\forest@ve{reversed}=0 +\e
7309 }
7310 \csdef{forest@anchor@@parent first'}{%
7311 \forest@anchor@isbordertrue
7312 \edef\forest@temp@anchor@parent{\number\numexpr\forest@ve{grow}-\forest@ve{rotate}+180}%
7313 \edef\forest@temp@anchor@first{\number\numexpr\forest@ve{grow}-\forest@ve{rotate}\ifnum\forest@ve{reversed}=
7314 \forest@getaverageangle{\forest@temp@anchor@parent}{\forest@temp@anchor@first}\forest@temp@anchor
7315 }
7316 \csdef{forest@anchor@@parent last'}{%
7317 \forest@anchor@isbordertrue
7318 \edef\forest@temp@anchor@parent{\number\numexpr\forest@ve{grow}-\forest@ve{rotate}+180}%
7319 \edef\forest@temp@anchor@last{\number\numexpr\forest@ve{grow}-\forest@ve{rotate}\ifnum\forest@ve{reversed}=
7320 \forest@getaverageangle{\forest@temp@anchor@parent}{\forest@temp@anchor@last}\forest@temp@anchor
7321 }
7322 \csdef{forest@anchor@@children first'}{%

```

```

7323 \forest@anchor@isbordertrue
7324 \edef\forest@temp@anchor@first{\number\numexpr\forestove{grow}-\forestove{rotate}\ifnum\forestove{reversed}=
7325 \forest@getaverageangle{\forestove{grow}-\forestove{rotate}}{\forest@temp@anchor@first}\forest@temp@anchor
7326 }
7327 \csdef{forest@anchor@@children last'}{%
7328 \forest@anchor@isbordertrue
7329 \edef\forest@temp@anchor@last{\number\numexpr\forestove{grow}-\forestove{rotate}\ifnum\forestove{reversed}=
7330 \forest@getaverageangle{\forestove{grow}-\forestove{rotate}}{\forest@temp@anchor@last}\forest@temp@anchor
7331 }
7332 \csdef{forest@anchor@@children}{%
7333 \forest@anchor@snapbordertocompasstrue
7334 \csuse{forest@anchor@@children'}%
7335 }
7336 \csdef{forest@anchor@@parent}{%
7337 \forest@anchor@snapbordertocompasstrue
7338 \csuse{forest@anchor@@parent'}%
7339 }
7340 \csdef{forest@anchor@@first}{%
7341 \forest@anchor@snapbordertocompasstrue
7342 \csuse{forest@anchor@@first'}%
7343 }
7344 \csdef{forest@anchor@@last}{%
7345 \forest@anchor@snapbordertocompasstrue
7346 \csuse{forest@anchor@@last'}%
7347 }
7348 \csdef{forest@anchor@@parent first}{%
7349 \forest@anchor@snapbordertocompasstrue
7350 \csuse{forest@anchor@@parent first'}%
7351 }
7352 \csdef{forest@anchor@@parent last}{%
7353 \forest@anchor@snapbordertocompasstrue
7354 \csuse{forest@anchor@@parent last'}%
7355 }
7356 \csdef{forest@anchor@@children first}{%
7357 \forest@anchor@snapbordertocompasstrue
7358 \csuse{forest@anchor@@children first'}%
7359 }
7360 \csdef{forest@anchor@@children last}{%
7361 \forest@anchor@snapbordertocompasstrue
7362 \csuse{forest@anchor@@children last'}%
7363 }

```

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>.

```

7364 \def\forest@getaverageangle#1#2#3{%
7365 \edef\forest@temp{\number\numexpr #1-#2+540}%
7366 \pgfmathMod{\forest@temp}{360}\pgfmathtruncatemacro\pgfmathresult{\pgfmathresult}
7367 \edef\forest@temp{360+#2+(\pgfmathresult-180)/2}%
7368 \pgfmathMod{\forest@temp}{360}\pgfmathtruncatemacro#3{\pgfmathresult}%
7369 }

```

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

```

7370 \def\forest@anchor@border@to@compass{%
7371 \ifforest@anchor@isborder
7372 \ifforest@anchor@snapbordertocompass
7373 \forest@anchor@snap@border@to@compass
7374 \else
7375 \pgfmathMod{\forest@temp@anchor}{360}%
7376 \pgfmathtruncatemacro\forest@temp@anchor{\pgfmathresult}%

```

```

7377 \fi
7378 \ifforest@anchor@forwardtotikz
7379 \ifcsname pgf@anchor%
7380 @\csname pgf@sh@ns@\pgfreferencednodename\endcsname
7381 @\csname forest@compass@\forest@temp@anchor\endcsname
7382 \endcsname
7383 \letcs\forest@temp@anchor{forest@compass@\forest@temp@anchor}%
7384 \fi
7385 \else
7386 \ifforest@anchor@snapbordertocompass
7387 \letcs\forest@temp@anchor{forest@compass@\forest@temp@anchor}%
7388 \fi
7389 \fi
7390 \fi
7391 }
7392 \csdef{forest@compass@0}{east}
7393 \csdef{forest@compass@45}{north east}
7394 \csdef{forest@compass@90}{north}
7395 \csdef{forest@compass@135}{north west}
7396 \csdef{forest@compass@180}{west}
7397 \csdef{forest@compass@225}{south west}
7398 \csdef{forest@compass@270}{south}
7399 \csdef{forest@compass@315}{south east}
7400 \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).

```

7401 \def\forest@anchor@snap@border@to@compass{%
7402 \pgfmathMod{\forest@temp@anchor}{360}%
7403 \pgfmathdivide{\pgfmathresult}{45}%
7404 \pgfmathround{\pgfmathresult}%
7405 \pgfmathmultiply{\pgfmathresult}{45}%
7406 \pgfmathtruncatemacro\forest@temp@anchor{\pgfmathresult}%
7407 }

```

This macro forwards the resulting anchor to TikZ.

```

7408 \def\forest@anchor@forward#1{% #1 = shape name
7409 \ifdefempty\forest@temp@anchor{%
7410 \pgf@sh@reanchor{#1}{center}%
7411 \xdef\forest@hack@tikzshapeborder{%
7412 \noexpand\tikz@shapebordertrue
7413 \def\noexpand\tikz@shapeborder@name{\pgfreferencednodename}%
7414 }\aftergroup\forest@hack@tikzshapeborder
7415 }{%
7416 \pgf@sh@reanchor{#1}{\forest@temp@anchor}%
7417 }%
7418 }

```

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

```

7419 \def\forest@referencednodeid{\forest@node@Nametoid{\forest@referencednodename}}%
7420 \def\forest@referencednodename{%
7421 \expandafter\expandafter\expandafter\forest@referencednodename@\expandafter\pgfreferencednodename\forest@pgf@notyetpositioned
7422 }
7423 \expandafter\def\expandafter\forest@referencednodename@\expandafter#\expandafter1\forest@pgf@notyetpositioned
7424 \if\relax#1\relax\forest@referencednodename@stripafter#2\relax\fi
7425 \if\relax#2\relax#1\fi
7426 }
7427 \expandafter\def\expandafter\forest@referencednodename@stripafter\expandafter#\expandafter1\forest@pgf@notyetpositioned

```

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


```

7428 \def\forest@Pointanchor#1#2{% #1 = node id, #2 = anchor
7429   {%
7430     \def\forest@pa@temp@name{name}%
7431     \forestOifdefined{#1}{@box}{%
7432       \forestOget{#1}{@box}\forest@temp
7433       \ifdefempty\forest@temp{%
7434         \def\forest@pa@temp@name{later@name}%
7435       }%
7436     }{%
7437       \setbox0\hbox{%
7438         \begin{pgfpicture}%
7439           \if\relax\forestOve{#1}{#2}\relax
7440             \pgfpointanchor{\forestOve{#1}{\forest@pa@temp@name}}{center}%
7441           \else
7442             \pgfpointanchor{\forestOve{#1}{\forest@pa@temp@name}}{\forestOve{#1}{#2}}%
7443           \fi
7444           \xdef\forest@global@marshal{%
7445             \noexpand\global\noexpand\pgf@x=\the\pgf@x\relax
7446             \noexpand\global\noexpand\pgf@y=\the\pgf@y
7447           }%
7448           \end{pgfpicture}%
7449         }%
7450       }%
7451       \forest@global@marshal
7452     }
7453 \def\forest@pointanchor#1{% #1 = anchor
7454   \forest@Pointanchor{\forest@cn}{#1}%
7455 }

```

10 Compatibility with previous versions

```

7456 \ifdefempty{\forest@compat}{%
7457   \RequirePackage{forest-compat}
7458 }

```