

作りましょう 0.7

パラメタ方式フォントファミリ  
校とプリティプリントのソース

Tsukurimashou 0.7

Parametric Font Family  
Proofs and pretty-printed  
source code

Matthew Skala

[mskala@ansuz.sooke.bc.ca](mailto:mskala@ansuz.sooke.bc.ca)

2013年3月7日

March 7, 2013

Proofs and pretty-printed source code for Tsukurimashou  
Copyright © 2011, 2012 Matthew Skala

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, version 3.

As a special exception, if you create a document which uses this font, and embed this font or unaltered portions of this font into the document, this font does not by itself cause the resulting document to be covered by the GNU General Public License. This exception does not however invalidate any other reasons why the document might be covered by the GNU General Public License. If you modify this font, you may extend this exception to your version of the font, but you are not obligated to do so. If you do not wish to do so, delete this exception statement from your version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>.

# Contents

I Infrastructure	13
preintro.mp	15
Infrastructure . . . . .	15
Font Parameter Defaults . . . . .	16
tsuku-bk.mp	18
tsuku-kg.mp	19
tsuku-mg.mp	20
tsuku-mi.mp	21
tsuku-ps.mp	22
tsuku-bq.mp	23
tsuku-dq.mp	24
tsuku-el.mp	25
tsuku-eq.mp	26
tsuku-lw.mp	27
intro.mp	28
fntbase.mp	46
General Library Functions . . . . .	46
Prefix And Suffix Handling . . . . .	53
A Module That Finds An Envelope Of A Path Drawn With A Pen . . . . .	55
Postscript Font Generation . . . . .	67
obstack.mp	89
Object Stack Data . . . . .	89
Object Stack Methods . . . . .	90
frac-intro.mp	95
latin-intro.mp	97
accent.mp	101
bcircle.mp	112
buildkanji.mp	115
dakuten.mp	131
enclosed.mp	132
genjimon.mp	134
hiragana.mp	146
Hiragana Vowels . . . . .	146
Hiragana Kakikukeko/Gagigugego . . . . .	151
Hiragana Sashisuseso/Zajizuzezo . . . . .	155
Hiragana Tachitsuteto/Dajizudedo . . . . .	161

Hiragana Naninuneno . . . . .	166
Hiragana Hahifuheho/Babibubebo/Papipupepo . . . . .	171
Hiragana Mamimumemo . . . . .	176
Hiragana Yayuyo . . . . .	182
Hiragana Rarirurero . . . . .	184
Hiragana Wawiwewo/N/Iteration . . . . .	190
iching.mp	198
katakana.mp	200
Katakana Vowels . . . . .	200
Katakana Kakikukeko/Gagigugego . . . . .	204
Katakana Sashisuseso/Zajizuzezo . . . . .	209
Katakana Tachitsuteto/Dajizudedo . . . . .	214
Katakana Naninuneno . . . . .	219
Katakana Hahifuheho/Babibubebo/Papipupepo . . . . .	224
Katakana Mamimumemo . . . . .	230
Katakana Yayuyo . . . . .	234
Katakana Rarirurero . . . . .	237
Katakana Wawiwewo/N/Iteration . . . . .	243
latin.mp	249
numerals.mp	338
ogonek.mp	349
punct.mp	361
serif.mp	403
 II Shared kyouiku kanji	 409
gradeone.mp	411
gradetwo.mp	478
gradethree.mp	551
gradefour.mp	590
gradefive.mp	619
gradesix.mp	631
 III Other shared kanji	 641
bottomrad.mp	643
leftrad.mp	646
radical.mp	654
rightrad.mp	689
toprad.mp	693
gradeeight.mp	698

gradenine.mp	732
rare.mp	759

IV U+0000 to U+2FFF	795
tsuku-00.mp	797
Ascii . . . . .	797
Latin-1 Extra Characters . . . . .	891
Accented Latin . . . . .	912
tsuku-01.mp	970
Latin Extended A Uppercase . . . . .	970
Latin Extended A Lowercase . . . . .	1025
Latin Extended A Other . . . . .	1083
tsuku-02.mp	1085
Latin Extended B . . . . .	1085
Spacing Modifier Letters . . . . .	1086
tsuku-03.mp	1095
Combining Diacritical Marks . . . . .	1095
tsuku-20.mp	1110
General Punctuation . . . . .	1110
tsuku-21.mp	1127
Symbols Required By Mes-1 . . . . .	1127
tsuku-24.mp	1133
Circled Numerals . . . . .	1133
Circled Latin And Zero . . . . .	1153
Inverted Circled Numerals . . . . .	1206
Doubly Circled Numerals . . . . .	1216
One More Inverted Circled Numeral . . . . .	1226
tsuku-25.mp	1228
Geometric Shapes . . . . .	1228
tsuku-26.mp	1230
I Ching . . . . .	1230
tsuku-27.mp	1247
Inverted Circled Numerals . . . . .	1247
tsuku-2e.mp	1258
Cjk Radicals Supplement . . . . .	1258
tsuku-2f.mp	1295
Kangxi Radicals . . . . .	1295
Ideographic Description Characters . . . . .	1415

V	U+3000 to U+4DFF	1429
	tsuku-30.mp	1431
	Ideographic Symbols And Punctuation . . . . .	1431
	Hiragana . . . . .	1451
	Katakana . . . . .	1496
	tsuku-31.mp	1543
	Phonetic Extensions For Ainu . . . . .	1543
	tsuku-32.mp	1560
	Circled Numerals . . . . .	1560
	Circled Katakana . . . . .	1590
	tsuku-34.mp	1638
	tsuku-4d.mp	1639
	I Ching . . . . .	1639
VI	U+4E00 to U+61FF	1705
	tsuku-4e.mp	1707
	tsuku-4f.mp	1744
	tsuku-50.mp	1798
	tsuku-51.mp	1808
	tsuku-52.mp	1820
	tsuku-53.mp	1850
	tsuku-54.mp	1873
	tsuku-55.mp	1885
	tsuku-56.mp	1891
	tsuku-57.mp	1898
	tsuku-58.mp	1905
	tsuku-59.mp	1911
	tsuku-5a.mp	1924
	tsuku-5b.mp	1926
	tsuku-5c.mp	1957
	tsuku-5d.mp	1981
	tsuku-5e.mp	1989
	tsuku-5f.mp	2024
	tsuku-60.mp	2069
	tsuku-61.mp	2118
VII	U+6200 to U+75FF	2125
	tsuku-62.mp	2127
	tsuku-63.mp	2143

tsuku-65.mp	2149
tsuku-66.mp	2165
tsuku-67.mp	2184
tsuku-68.mp	2199
tsuku-69.mp	2206
tsuku-6a.mp	2212
tsuku-6b.mp	2216
tsuku-6c.mp	2225
tsuku-6d.mp	2275
tsuku-6e.mp	2288
tsuku-6f.mp	2299
tsuku-70.mp	2304
tsuku-71.mp	2309
tsuku-72.mp	2311
tsuku-73.mp	2320
tsuku-74.mp	2321
tsuku-75.mp	2325

VIII U+7600 to U+89FF	2337
tsuku-76.mp	2339
tsuku-77.mp	2346
tsuku-78.mp	2356
tsuku-79.mp	2358
tsuku-7a.mp	2369
tsuku-7b.mp	2376
tsuku-7c.mp	2389
tsuku-7d.mp	2392
tsuku-7f.mp	2408
tsuku-80.mp	2413
tsuku-81.mp	2421
tsuku-82.mp	2426
tsuku-83.mp	2457
tsuku-84.mp	2469
tsuku-85.mp	2474
tsuku-86.mp	2476
tsuku-88.mp	2477
tsuku-89.mp	2482

IX	U+8A00 to U+9FFF	2487
	tsuku-8a.mp	2489
	tsuku-8b.mp	2544
	tsuku-8c.mp	2547
	tsuku-8d.mp	2555
	tsuku-8e.mp	2559
	tsuku-8f.mp	2564
	tsuku-90.mp	2576
	tsuku-91.mp	2595
	tsuku-92.mp	2606
	tsuku-93.mp	2610
	tsuku-95.mp	2613
	tsuku-96.mp	2630
	tsuku-97.mp	2652
	tsuku-98.mp	2656
	tsuku-99.mp	2667
	tsuku-9a.mp	2671
	tsuku-9b.mp	2675
	tsuku-9c.mp	2676
	tsuku-9e.mp	2678
	tsuku-9f.mp	2681
X	U+A000 to U+10FFFF	2683
	tsuku-f7.mp	2685
	Latin Small Caps . . . . .	2685
	tsuku-ff.mp	2712
	Full-Width Forms . . . . .	2712
	Half-Width Punctuation . . . . .	2738
	Half-Width Katakana . . . . .	2741
	tsuku-1f1.mp	2805
	Squared Latin . . . . .	2805
	Inverse Circled Latin . . . . .	2831
	Inverse Squared Latin . . . . .	2857
	tsuku-200.mp	2884
	tsuku-21c.mp	2885
	tsuku-f17.mp	2886
	Combining Dots For I Ching . . . . .	2886
	Miscellaneous . . . . .	2895
	Tomoe Ornaments . . . . .	2901

Heavy Metal Umlaut . . . . .	2909
Genjimon . . . . .	2926
tsuku-ff0.mp . . . . .	2980
Fraction Numerators . . . . .	2980
tsuku-ff1.mp . . . . .	2987
Fraction Denominators . . . . .	2987
 XI Jieubsida core . . . . .	 2989
hangul.mp . . . . .	2991
Jamo Combining Operations . . . . .	2993
jamo-basic.mp . . . . .	3001
Filler Jamo . . . . .	3001
Sios/Cieuc/Chieuch Family . . . . .	3001
Kiyeok . . . . .	3004
Nieun . . . . .	3005
Tikeut . . . . .	3007
Rieul . . . . .	3009
Mieum . . . . .	3010
Pieup . . . . .	3011
Sios . . . . .	3012
Ieung . . . . .	3012
Cieuc . . . . .	3013
Chieuch . . . . .	3014
Khieukh . . . . .	3015
Thieuth . . . . .	3016
Phieuph . . . . .	3017
Hieuh . . . . .	3019
Mixed Tails . . . . .	3020
Vowels . . . . .	3021
jamo-extra.mp . . . . .	3030
Pansios . . . . .	3033
Yesieung . . . . .	3033
Yeorinhieuh . . . . .	3034
Chitueum And Ceongchieum Variants . . . . .	3036
Kapyeoun Variants . . . . .	3037
hgllextb.mp . . . . .	3038
Hangul Extension B . . . . .	3038
Hangul Jungseong (Vowel) Jamo Extension B . . . . .	3038
Hangul Jongseong (Tail) Jamo Extension B . . . . .	3061

jieub-bt.mp	3111
jieub-do.mp	3112
jieub-sm.mp	3113
hglpage.mp	3114
jieub-11.mp	3116
Hangul Choseong (Lead) Jamo . . . . .	3116
Hangul Jungseong (Vowel) Jamo . . . . .	3211
Hangul Jongseong (Tail) Jamo . . . . .	3282
jieub-31.mp	3371
Hangul Compatibility Jamo . . . . .	3371
jieub-a9.mp	3465
Hangul Choseong (Lead) Jamo Extended A . . . . .	3465
jieub-ac.mp	3495
 XII Jieubsida alternates	 3549
jieub-ff2.mp	3551
Hangul Jungseong (Vowel) Jamo . . . . .	3551
Hangul Jungseong (Vowel) Jamo Extension B . . . . .	3622
jieub-ff3.mp	3646
Hangul Choseong (Lead) Jamo . . . . .	3646
jieub-ff4.mp	3771
Hangul Choseong (Lead) Jamo . . . . .	3771
jieub-ff5.mp	3896
Hangul Choseong (Lead) Jamo . . . . .	3896
jieub-ff6.mp	4021
Hangul Choseong (Lead) Jamo . . . . .	4021
jieub-ff7.mp	4146
Hangul Choseong (Lead) Jamo . . . . .	4146
Hangul Choseong (Lead) Jamo Extended A . . . . .	4241
 XIII TsuIta	 4271
tsuita-common.mp	4273
tsuita-at.mp	4284
tsuita-so.mp	4285
Additional Proofs	4287
 XIV Blackletter Lolita	 4341
bll.mp	4343

bll-co.mp	4348
pentacross.mp	4349
Utilities For Pentagrams And Crosses . . . . .	4349
bll-f5c.mp	4351
Pentagrams . . . . .	4351



# Volume I

## Infrastructure

preintro.mp	<b>PRE</b>
tsuku-bk.mp	<b>BK</b>
tsuku-kg.mp	<b>KG</b>
tsuku-mg.mp	<b>MG</b>
tsuku-mi.mp	<b>MI</b>
tsuku-ps.mp	<b>PS</b>
tsuku-bq.mp	<b>BQ</b>
tsuku-dq.mp	<b>DQ</b>
tsuku-el.mp	<b>EL</b>
tsuku-eq.mp	<b>EQ</b>
tsuku-lw.mp	<b>LW</b>
intro.mp	<b>INTR</b>
fntbase.mp	<b>FNTB</b>
obstack.mp	<b>OBST</b>
frac-intro.mp	<b>FRAC</b>
latin-intro.mp	<b>LATI</b>
accent.mp	<b>ACCE</b>
bcircle.mp	<b>BCIR</b>
buildkanji.mp	<b>BUIL</b>
dakuten.mp	<b>DAKU</b>
enclosed.mp	<b>ENCL</b>
genjimon.mp	<b>GENJ</b>
hiragana.mp	<b>HIRA</b>
iching.mp	<b>ICHI</b>
katakana.mp	<b>KATA</b>
latin.mp	<b>LATI</b>
numerals.mp	<b>NUME</b>
ogonek.mp	<b>OGON</b>
punct.mp	<b>PUNC</b>
serif.mp	<b>SERI</b>



```

1 %
2 % Early shared code for Tsukurimashou
3 % Copyright (C) 2011, 2012 Matthew Skala
4 %
5 % This program is free software: you can redistribute it and/or modify
6 % it under the terms of the GNU General Public License as published by
7 % the Free Software Foundation, version 3.
8 %
9 % As a special exception, if you create a document which uses this font, and
10 % embed this font or unaltered portions of this font into the document, this
11 % font does not by itself cause the resulting document to be covered by the
12 % GNU General Public License. This exception does not however invalidate any
13 % other reasons why the document might be covered by the GNU General Public
14 % License. If you modify this font, you may extend this exception to your
15 % version of the font, but you are not obligated to do so. If you do not
16 % wish to do so, delete this exception statement from your version.
17 %
18 % This program is distributed in the hope that it will be useful,
19 % but WITHOUT ANY WARRANTY; without even the implied warranty of
20 % MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
21 % GNU General Public License for more details.
22 %
23 % You should have received a copy of the GNU General Public License
24 % along with this program. If not, see <http://www.gnu.org/licenses/>.
25 %
26 % Matthew Skala
27 % http://ansuz.sooke.bc.ca/
28 % mskala@ansuz.sooke.bc.ca
29 %
30
31 _____
32

```

## Infrastructure

```

33 % INFRASTRUCTURE
34
35 % When I say nonstopmode I mean nonstopmode, dammit!
36 nonstopmode;
37 def errorstopmode = nonstopmode enddef;
38
39 % no chars we don't define, please
40 no_implicit_spaces:=1;
41
42 % load library from METATYPE1
43 input fntbase.mp;
44

```

```

45 % file inclusion gatekeeper
46 vardef inclusion_lock(suffix fn) =
47   if known_already_included.fn:
48     endinput;
49   fi;
50   boolean already_included.fn;
51   already_included.fn:=true;
52 enddef;
53
54 % late inclusion
55 numeric late_include_count;
56 late_include_count:=0;
57 string late_include[];
58
59 vardef include_late(expr fn) =
60   late_include_count:=late_include_count+1;
61   late_include[late_include_count]:=fn;
62 enddef;
63
64 vardef do_late_includes =
65   for i:=1 upto late_include_count:
66     scantokens ("input " & late_include[i]);
67   endfor;
68   late_include_count:=0;
69 enddef;
70
71 _____
72

```

## Font Parameter Defaults

```

73 % FONT PARAMETER DEFAULTS
74
75 % basic brush definition
76 transform tsu_brush_xf;
77 tsu_brush_shape:=1.0;
78 tsu_brush_angle:=0;
79
80 % special brush for punctuation
81 tsu_pbrush_size:=50;
82 tsu_pbrush_shape:=1.0;
83 tsu_pbrush_angle:=0;
84 tsu_punct_size:=100;
85
86 % size the handakuten
87 handakuten_inner:=120;
88 handakuten_outer:=200;
89

```

```

90 % general shape tweaker
91 mincho:=0;
92
93 % slant during rescaling, for italics
94 rescale_slant:=0;
95
96 % control appearance of corners
97 boolean sharp_corners;
98 sharp_corners:=false;
99
100 % for naming the font
101 string familyname,stylename;
102 familyname:="Tsukurimashou";
103 stylename:="";
104
105 % brush option override
106 def tsu_brush_opt(expr n,l) = nib(n)(l) enddef;
107
108 % bo_serif type; point lp; direction lp; brush tip size
109 vardef tsu_serif.choose(expr bst,plp,dlp,l,bts,bos) =
110 enddef;
111
112 % do "modern" width alternation
113 boolean do_alteration;
114 do_alteration:=false;
115
116 % handle outline mode for Genjimon
117 boolean genji_outline;
118 genji_outline:=false;
119
120 % prepare to detect proportional spacing
121 boolean is_proportional;
122 is_proportional:=false;
123
124 % prepare to detect blackletter
125 boolean is_blackletter;
126 is_blackletter:=false;
127
128 % prepare to detect fine IDCs
129 boolean fine_idcs;
130 fine_idcs:=false;
131
132 % prepare to detect italic hook shapes
133 boolean do_italic_hook;
134 do_italic_hook:=false;

```

## tsuku-bk.mp

```

1 %
2 % Tsukurimashou Bokukko
3 % Copyright (C) 2011 Matthew Skala
4 %
5-29 [Standard copyright notice]
30
31 % TSUKURIMASHOU BOKUKKO
32
33 input preintro.mp;
34
35 stylename:="Bokukko";
36
37 mincho:=0.3;
38
39 (0,4) transformed tsu_brush_xf = (0.8,0.95);
40 (1,1) transformed tsu_brush_xf = (1.02,0.80);
41 (4,0) transformed tsu_brush_xf = (3.8,0.95);
42
43 tsu_brush_min:=0.80;
44 tsu_brush_max:=0.95;
45 tsu_brush_shape:=0.3;
46 tsu_brush_angle:=20;
47
48 tsu_pbrush_size:=60;
49 tsu_pbrush_shape:=0.3;
50 tsu_pbrush_angle:=20;
51
52 def tsu_brush_opt(expr n,l) = cut(n,rel 120)(l) enddef;
53 sharp_corners:=true;
54
55 genji_outline:=true;
56 genji_hw:=0.55;
57
58 input intro.mp;

```

## tsuku-kg.mp

```
1 %
2 % Tsukurimashou Kaku
3 % Copyright (C) 2011 Matthew Skala
4 %
5-29 [Standard copyright notice]
30
31 % TSUKURIMASHOU KAKU
32
33 input preintro.mp;
34
35 stylename:="Kaku";
36
37 (0,4) transformed tsu_brush_xf = (4,0.75);
38 (1,1) transformed tsu_brush_xf = (1,0.62);
39 (4,0) transformed tsu_brush_xf = (0,0.75);
40
41 tsu_brush_min:=0.62;
42 tsu_brush_max:=0.75;
43
44 def tsu_brush_opt(expr n,l) = cut(n,rel 90)(l) enddef;
45 sharp_corners:=true;
46
47 input intro.mp;
```

## tsuku-mg.mp

MG

```
1 %
2 % Tsukurimashou Maru
3 % Copyright (C) 2011 Matthew Skala
4 %
5-29 [Standard copyright notice]
30
31 % TSUKURIMASHOU MARU
32
33 input preintro.mp;
34
35 stylename:="Maru";
36
37 (0,4) transformed tsu_brush_xf = (4,0.74);
38 (1,1) transformed tsu_brush_xf = (1,0.65);
39 (4,0) transformed tsu_brush_xf = (0,0.74);
40
41 tsu_brush_min:=0.65;
42 tsu_brush_max:=0.74;
43
44 input intro.mp;
```

## tsuku-mi.mp

```
1 %
2 % Tsukurimashou Mincho
3 % Copyright (C) 2011 Matthew Skala
4 %
5-29 [Standard copyright notice]
30
31 % TSUKURIMASHOU MINCHO
32
33 input preintro.mp;
34
35 stylename:="Mincho";
36
37 mincho:=1;
38
39 (0,4) transformed tsu_brush_xf = (0,0,1,1);
40 (1,1) transformed tsu_brush_xf = (1,2,0,35);
41 (4,0) transformed tsu_brush_xf = (4,8,1,1);
42
43 tsu_brush_min:=0.35;
44 tsu_brush_max:=1.05;
45
46 tsu_brush_shape:=0.38;
47 tsu_brush_angle:=1;
48
49 tsu_pbrush_size:=60;
50 tsu_pbrush_shape:=0.38;
51 tsu_pbrush_angle:=1;
52
53 input serif.mp;
54
55 for i=1 upto 10:
56   tsu_do_serif[i]:=true;
57 endfor;
58
59 do_alteration:=true;
60
61 genji_outline:=true;
62 genji_hw:=0.2;
63
64 input intro.mp;
```

## tsuku-ps.mp

```
1 %
2 % Proportional spacing modifications for Tsukurimashou
3 % Copyright (C) 2011 Matthew Skala
4 %
5-29 [Standard copyright notice]
30
31 _____
32
33 vardef tsu_rescale_half = tsu_rescale_full; enddef;
34 vardef tsu_rescale_half_lc = tsu_rescale_full; enddef;
35 vardef tsu_rescale_half_katakana = tsu_rescale_full; enddef;
36 vardef tsu_rescale_decenter = tsu_rescale_full; enddef;
37
38 is_proportional:=true;
39
40 tsu_rescale_full;
```

PS

## tsuku-bq.mp

```
1 %
2 % Bold font weight
3 % Copyright (C) 2012 Matthew Skala
4 %
5-29 [Standard copyright notice]
30
31 % BOLD WEIGHT
32
33 transform weight_xf;
34 (0,0.62) transformed weight_xf = (0,1.082);
35 (1,0.62) transformed weight_xf = (1,1.082);
36 (0,0.75) transformed weight_xf = (0,1.191);
37
38 tsu_brush_xf:=tsu_brush_xf transformed weight_xf;
39 tsu_brush_min:=ypart ((0,tsu_brush_min) transformed weight_xf);
40 tsu_brush_max:=ypart ((0,tsu_brush_max) transformed weight_xf);
41
42 tsu_pbrush_size:=tsu_pbrush_size*(115/100);
43 tsu_punct_size:=120;
44
45 handakuten_outer:=260;
46
47 calc_mbrush_size;
```

**BQ**

## tsuku-dq.mp

```
1 %
2 % Demibold font weight
3 % Copyright (C) 2012 Matthew Skala
4 %
5-29 [Standard copyright notice]
30
31 % DEMIBOLD WEIGHT
32
33 transform weight_xf;
34 (0,0.62) transformed weight_xf = (0,0.781);
35 (1,0.62) transformed weight_xf = (1,0.781);
36 (0,0.75) transformed weight_xf = (0,0.945);
37
38 tsu_brush_xf:=tsu_brush_xf transformed weight_xf;
39 tsu_brush_min:=ypart ((0,tsu_brush_min) transformed weight_xf);
40 tsu_brush_max:=ypart ((0,tsu_brush_max) transformed weight_xf);
41
42 tsu_pbrush_size:=tsu_pbrush_size*(115/100);
43 tsu_punct_size:=110;
44
45 handakuten_outer:=260;
46
47 calc_mbrush_size;
```

## tsuku-el.mp

```
1 %
2 % Extra-Light font weight (Tenshi no Kami when added to Maru)
3 % Copyright (C) 2011, 2012 Matthew Skala
4 %
5-29 [Standard copyright notice]
30
31 % EXTRA-LIGHT WEIGHT
32
33 transform weight_xf;
34 (0,0.62) transformed weight_xf = (0,0.15);
35 (1,0.62) transformed weight_xf = (1,0.15);
36 (0,0.75) transformed weight_xf = (0,0.15);
37
38 tsu_brush_xf:=tsu_brush_xf transformed weight_xf;
39 tsu_brush_min:=ypart ((0,tsu_brush_min) transformed weight_xf);
40 tsu_brush_max:=ypart ((0,tsu_brush_max) transformed weight_xf);
41
42 tsu_pbrush_size:=tsu_pbrush_size*(15/100);
43 tsu_punct_size:=80;
44
45 handakuten_inner:=170;
46
47 fine_idcs:=true;
48
49 calc_mbrush_size;
```

EL

## tsuku-eq.mp

```
1 %
2 % Extra-Bold font weight (Anbiruteki when added to Maru)
3 % Copyright (C) 2011, 2012 Matthew Skala
4 %
5-29 [Standard copyright notice]
30
31 % EXTRA-BOLD WEIGHT
32
33 transform weight_xf;
34 (0,0.62) transformed weight_xf = (0,1.5);
35 (1,0.62) transformed weight_xf = (1,1.5);
36 (0,0.75) transformed weight_xf = (0,1.5);
37
38 tsu_brush_xf:=tsu_brush_xf transformed weight_xf;
39 tsu_brush_min:=ypart ((0,tsu_brush_min) transformed weight_xf);
40 tsu_brush_max:=ypart ((0,tsu_brush_max) transformed weight_xf);
41
42 tsu_pbrush_size:=tsu_pbrush_size*(115/100);
43 tsu_punct_size:=130;
44
45 handakuten_outer:=260;
46
47 calc_mbrush_size;
```

EQ

## tsuku-lw.mp

```
1 %
2 % Light font weight
3 % Copyright (C) 2012 Matthew Skala
4 %
5-29 [Standard copyright notice]
30
31 % LIGHT WEIGHT
32
33 transform weight_xf;
34 (0,0.62) transformed weight_xf = (0,0.386);
35 (1,0.62) transformed weight_xf = (1,0.386);
36 (0,0.75) transformed weight_xf = (0,0.439);
37
38 tsu_brush_xf:=tsu_brush_xf transformed weight_xf;
39 tsu_brush_min:=ypart ((0,tsu_brush_min) transformed weight_xf);
40 tsu_brush_max:=ypart ((0,tsu_brush_max) transformed weight_xf);
41
42 tsu_pbrush_size:=tsu_pbrush_size*(35/100);
43 tsu_punct_size:=90;
44
45 handakuten_inner:=170;
46
47 fine_idcs:=true;
48
49 calc_mbrush_size;
```

LW

## intro.mp

```
1 %
2 % General shared code for Tsukurimashou
3 % Copyright (C) 2011, 2012 Matthew Skala
4 %
5-29 [Standard copyright notice]
30
31 inclusion_lock(intro);
32
33 

---


34
35 %
36 % Tsukurimashou intro - utility routines for all pages
37 %
38
39 slang:=0;
40
41 pf_info_quad 1000;
42 pf_info_space 1000, 0, 0;
43 pf_info_familyname familyname;
44 pf_info_fontname
45   (familyname & stylename & "Subfont"),
46   (familyname & " " & stylename & " Subfont");
47 pf_info_fixedpitch true;
48 pf_info_capheight 900;
49 pf_info_xheight 585;
50 pf_info_ascender 985;
51 pf_info_descender 265;
52
53 pair centre_pt;
54 centre_pt=(500,390);
55 latin_vcentre:=430;
56 latin_wide_baseline:=25;
57 latin_wide_top:=750;
58 wide_margin:=30;
59 narrow_margin:=40;
60
61 

---


62
63 input bcircle.mp;
64 input obstack.mp;
65
66 

---


67
68 default_nib:=fix_nib(100,100,0);
69
70 def begintsuglyph(expr name,code) =
```

**INTR**

```

71 message name;
72 encode(name) (code); standard_introduce(name);
73 write ("BEGINGLYPH ""&name& "" "&decimal code) to "proof.prf";
74 beginglyph(name);
75   init_obstack;
76   string perl_structure;
77   perl_structure="$structure{""&name&"}=["""&name&""";
78 enddef;
79
80 def endtsuglyph =
81   if rescale_to.right>0:
82     fix_hsbw((rescale_to.left+rescale_to.right),0,0);
83   else:
84     fix_hsbw(0,0,0);
85   fi;
86 endglyph;
87 perl_structure:=perl_structure&"]";
88 write "PERL_STRUCTURE "&perl_structure to "proof.prf";
89 if rescale_to.right>0:
90   write ("ENDGLYPH 0 "&decimal (rescale_to.left+rescale_to.right))
91     to "proof.prf";
92 else:
93   write "ENDGLYPH -10 0" to "proof.prf";
94 fi;
95 sp:=0;
96 enddef;
97
98 def tsu_brush_tip_size(expr l,q) =
99   begingroup
100     numeric yy;
101     y:=ypart (point l of q);
102     if y<tsu_brush_min:
103       yy:=tsu_brush_min;
104     elseif y>tsu_brush_max:
105       yy:=tsu_brush_max;
106     else:
107       yy:=y;
108     fi;
109     yy
110   endgroup
111 enddef;
112
113 def tsu_brush_tip(expr l,p,q,bsi,is_start,is_end,is_alt) =
114   begingroup
115     numeric y;
116     y=tsu_brush_tip_size(l,q);
117     if is_alt and do_alteration:
118       fix_nib(bsi*y*tsu_brush_shape,bsi*y*tsu_brush_shape,tsu_brush_angle)

```

```

119     else:
120         fix_nib(bsi*y,bsi*y*tsu_brush_shape,tsu_brush_angle)
121     fi
122 endgroup
123 enddef;
124
125 % rescaling for half/double width
126 % this is basically global because it will be shared by most glyphs in a file
127
128 path width_curve;
129
130 def tsu_rescale_full =
131     rescale_from.left:=wide_margin;
132     rescale_from.right:=1000-wide_margin;
133     rescale_from.top:=ypart centre_pt;
134     rescale_from.bottom:=latin_wide_baseline;
135
136     rescale_to.left:=wide_margin;
137     rescale_to.right:=1000-wide_margin;
138     rescale_to.top:=ypart centre_pt;
139     rescale_to.bottom:=latin_wide_baseline;
140
141     rescale_skew:=0;
142
143     width_curve:=(-1,1)-(2000,2000);
144 enddef;
145
146 def tsu_rescale_half =
147     rescale_from.left:=wide_margin;
148     rescale_from.right:=1000-wide_margin;
149     rescale_from.top:=ypart centre_pt;
150     rescale_from.bottom:=latin_wide_baseline;
151
152     rescale_to.left:=narrow_margin;
153     rescale_to.right:=500-narrow_margin;
154     rescale_to.top:=latin_vcentre;
155     rescale_to.bottom:=0;
156
157     rescale_skew:=0;
158
159     width_curve:=((-1,1)-(100,100)).(940,410)..{right}(2000,1000);
160 enddef;
161
162 def tsu_rescale_half_lc =
163     rescale_from.left:=wide_margin*3.5;
164     rescale_from.right:=1000-wide_margin*3.5;
165     rescale_from.top:=ypart centre_pt;
166     rescale_from.bottom:=latin_wide_baseline;

```

```

167
168 rescale_to.left:=narrow_margin;
169 rescale_to.right:=500-narrow_margin;
170 rescale_to.top:=latin_vcentre;
171 rescale_to.bottom:=0;
172
173 rescale_skew:=0;
174
175 width_curve:=((-1,-1)-(100,100))..(780,410)..{right}(2000,1000);
176 enddef;
177
178 def tsu_rescale_half_katakana =
179   rescale_from.left:=wide_margin;
180   rescale_from.right:=1000-wide_margin;
181   rescale_from.top:=700;
182   rescale_from.bottom:=0;
183
184   rescale_to.left:=narrow_margin;
185   rescale_to.right:=500-narrow_margin;
186   rescale_to.top:=670;
187   rescale_to.bottom:=30;
188
189   rescale_skew:=8;
190
191   width_curve:=((-1,-1)-(100,100))..(860,440)..{right}(2000,1000);
192 enddef;
193
194 def tsu_rescale_double =
195   rescale_from.left:=narrow_margin;
196   rescale_from.right:=500-narrow_margin;
197   rescale_from.top:=latin_vcentre;
198   rescale_from.bottom:=0;
199
200   rescale_to.left:=wide_margin;
201   rescale_to.right:=1000-wide_margin;
202   rescale_to.top:=ypart centre_pt;
203   rescale_to.bottom:=latin_wide_baseline;
204
205   rescale_skew:=0;
206
207   width_curve:=(-1,-1)-(2000,2000);
208 enddef;
209
210 def tsu_rescale_decenter =
211   rescale_from.left:=300;
212   rescale_from.right:=700;
213   rescale_from.top:=ypart centre_pt;
214   rescale_from.bottom:=latin_wide_baseline;

```

```

215
216 rescale_to.left:=50;
217 rescale_to.right:=450;
218 rescale_to.top:=latin_vcentre;
219 rescale_to.bottom:=0;
220
221 rescale_skew:=0;
222
223 width_curve:=(-1,1)-(2000,2000);
224 enddef;
225
226 def tsu_rescale_native_narrow =
227   rescale_from.left:=narrow_margin;
228   rescale_from.right:=500-narrow_margin;
229   rescale_from.top:=latin_vcentre;
230   rescale_from.bottom:=0;
231
232   rescale_to.left:=narrow_margin;
233   rescale_to.right:=500-narrow_margin;
234   rescale_to.top:=latin_vcentre;
235   rescale_to.bottom:=0;
236
237   rescale_skew:=0;
238
239   width_curve:=(-1,1)-(2000,2000);
240 enddef;
241
242 def tsu_rescale_native_zero =
243   rescale_from.left:=0;
244   rescale_from.right:=0;
245   rescale_from.top:=1000;
246   rescale_from.bottom:=0;
247
248   rescale_to.left:=0;
249   rescale_to.right:=0;
250   rescale_to.top:=1000;
251   rescale_to.bottom:=0;
252
253   rescale_skew:=0;
254
255   width_curve:=(-1,1)-(2000,2000);
256 enddef;
257
258 def tsu_rescale_native_conditional =
259   if is_proportional:
260     tsu_rescale_full;
261   else:
262     tsu_rescale_native_narrow;

```

```

263 fi;
264 enddef;
265
266 def tsu_rescale_combining_full =
267   rescale_from.left:=wide_margin;
268   rescale_from.right:=1000-wide_margin;
269   rescale_from.top:=ypart centre_pt;
270   rescale_from.bottom:=latin_wide_baseline;
271
272   rescale_to.left:=wide_margin-1000;
273   rescale_to.right:=-wide_margin;
274   rescale_to.top:=ypart centre_pt;
275   rescale_to.bottom:=latin_wide_baseline;
276
277   rescale_skew:=0;
278
279   width_curve:=(-1,-1)-(2000,2000);
280 enddef;
281
282 def tsu_rescale_combining_half =
283   rescale_from.left:=wide_margin;
284   rescale_from.right:=1000-wide_margin;
285   rescale_from.top:=ypart centre_pt;
286   rescale_from.bottom:=latin_wide_baseline;
287
288   rescale_to.left:=narrow_margin-500;
289   rescale_to.right:=-narrow_margin;
290   rescale_to.top:=latin_vcentre;
291   rescale_to.bottom:=0;
292
293   rescale_skew:=0;
294
295   width_curve:=((-1,-1)-(100,100)).(940,410)..{right}(2000,1000);
296 enddef;
297
298 def tsu_rescale_combining_accent =
299   if is_proportional:
300     tsu_rescale_combining_full;
301   else:
302     tsu_rescale_combining_half;
303   fi;
304 enddef;
305
306 tsu_rescale_full;
307
308 def tsu_slant_xform =
309   begingroup
310     save st,cp;

```

```

311 transform st;
312 pair cp;
313
314 cp:=((rescale_from.left+rescale_from.right)/2,rescale_from.bottom);
315 cp transformed st=cp;
316 cp+(100,0) transformed st=cp+(100,0);
317 cp+(0,100) transformed st=cp+(rescale_slant/10,100);
318 st
319 endgroup
320 enddef;
321
322 def tsu_rescale_xform =
323   begingroup
324     save t,st,cp;
325     transform t,st;
326     st:=tsu_slant_xform;
327     t:=st;
328     % check if rescaling is active
329     if (rescale_from.left<>rescale_to.left)
330     or (rescale_from.right<>rescale_to.right): begingroup
331       save i,xa,xb,lf,rf,wf,lt,rt,wt;
332       numeric i,xa,xb,lf,rf,wf,lt,rt,wt;
333       transform t;
334       % find the bounds of the paths
335       if find_stroke(0)<=0:
336         xa:=0.5[rescale_from.left,rescale_from.right];
337         xb:=0.5[rescale_from.left,rescale_from.right];
338       else:
339         xa:=infinity;
340         xb:=-infinity;
341         for i=1 upto sp-1:
342           if obstacktype[i]=otstroke:
343             if (xpart llcorner obstackp[i])<xa:
344               xa:=xpart llcorner obstackp[i];
345             fi;
346             if (xpart lrcorner obstackp[i])>xb:
347               xb:=xpart lrcorner obstackp[i];
348             fi;
349           fi;
350         endfor;
351       fi;
352       % compute bearings and widths
353       lf=xa-rescale_from.left;
354       rf=rescale_from.right-xb;
355       lf+rf+wf=rescale_from.right-rescale_from.left;
356       lt+rt+wt=rescale_to.right-rescale_to.left;
357       (lt,rt)=whatever[(0,0),(lf,rf)];
358       wt=ypart (width_curve intersectionpoint ((wf,-infinity)-(wf,infinity)));

```

```

359 % find transformation
360 if wf>0:
361     (rescale_from.left+lf,rescale_from.bottom) transformed t=
362     (rescale_to.left+lt,rescale_to.bottom-rescale_skew);
363     (rescale_from.left+lf,rescale_from.top) transformed t=
364     (rescale_to.left+lt,rescale_to.top-rescale_skew);
365     (rescale_from.right-rf,rescale_from.bottom) transformed t=
366     (rescale_to.right-rt,rescale_to.bottom+rescale_skew);
367 else:
368     (rescale_from.left+lf,rescale_from.bottom) transformed t=
369     (rescale_to.left+lt,rescale_to.bottom);
370     (rescale_from.left+lf,rescale_from.top) transformed t=
371     (rescale_to.left+lt,rescale_to.top);
372     (rescale_from.left+lf+1,rescale_from.bottom) transformed t=
373     (rescale_to.left+lt+1,rescale_to.bottom);
374 fi;
375 pair cp;
376 transform st;
377 cp:=((rescale_to.left+rescale_to.right)/2,rescale_to.bottom);
378 cp transformed st=cp;
379 cp+(100,0) transformed st=cp+(100,0);
380 cp+(0,100) transformed st=cp+(rescale_slant/10,100);
381 t:=t transformed st;
382 endgroup; fi;
383 t
384 endgroup
385 enddef;
386
387 % solve the quadratic equation  $ax^2+bx+c=0$ , including pathological cases
388 vardef solve_quadratic(expr a,b,c) =
389   if a=0:
390     if b=0:
391       if c=0:
392         (0,whatever)
393       else:
394         (whatever,whatever)
395     fi
396   else:
397     (-c/b,whatever)
398   fi
399 elseif abs(a)<abs(b)/500:
400   (whatever,whatever)
401 else:
402   save d;
403   numeric d;
404   d=b*b-4*a*c;
405   if d>0:
406     ((-b-sqrt(d),-b+sqrt(d))/(2*a))

```

```

407 elseif d=0:
408     (-b/(2*a),whatever)
409 else:
410     (whatever,whatever)
411 fi
412 fi
413 enddef;
414
415 % find the t-values of any inflection points of subpath (0,1) of p
416 vardef segment_inflections(expr p) =
417     save x,y,c;
418     numeric x[],y[],c[];
419
420     % normalize to prevent numerical misbehaviour
421     z10=(point 0 of p)+z22;
422     z11=(postcontrol 0 of p)+z22;
423     z12=(precontrol 1 of p)+z22;
424     z13=(point 1 of p)+z22;
425     z10+z11+z12+z13=(0,0);
426     c10=(abs(z10)+abs(z11)+abs(z12)+abs(z13))/4;
427     if c10<0.1:
428         c10:=0.1;
429     fi;
430     z0=z10/c10;
431     z1=z11/c10;
432     z2=z12/c10;
433     z3=z13/c10;
434
435     % abort if points are close enough to collinear
436     if (abs(x0*y1-x1*y0)<0.01) and (abs(x2*y3-x3*y2)<0.01):
437         (whatever,whatever)
438     else:
439
440         % find t-values at which |z'(t) cross z''(t)|=0
441         c2=y0*( -1*x1 +2*x2 -x3)
442             +y1*( x0 -3*x2 +2*x3)
443             +y2*(-2*x0 +3*x1 -x3)
444             +y3*( x0 -2*x1 +x2 );
445
446         c1=y0*( 2*x1 -3*x2 +x3)
447             +y1*(-2*x0 +3*x2 -x3)
448             +y2*( 3*x0 -3*x1 )
449             +y3*( -x0 +x1 );
450
451         c0=y0*( -x1 +x2)
452             +y1*( x0 -x2)
453             +y2*( -x0 +x1 );
454

```

```

455     z20=solve_quadratic(c2,c1,c0);
456
457     % filter and sort to find points properly within the segment
458     if known x20:
459         if (x20>0.01) and (x20<0.99):
460             x21=x20;
461             fi;
462         fi;
463     if known y20:
464         if (y20>0.01) and (y20<0.99):
465             y21=y20;
466             fi;
467         fi;
468     if known x21:
469         z21
470     else:
471         (y21,x21)
472     fi
473 fi
474 enddef;
475
476 % version of insert_nodes modified to *always* insert, which is needed
477 % to keep node numbers in sync on pen-size-control paths
478 vardef really_insert_nodes(expr p)(text t) =
479     save j_, p_, s_, t_; path p_; p_:=p;
480     t_:=0;
481     for i_:=t:
482         t_[incr t_]=arclength(subpath(0,i_ mod length(p_)) of p_);
483     endfor
484     for i_:=1 upto t_:
485         s_:=arctime t_[i_] of p_;
486         p_=(subpath (0, s_) of p_) && (subpath (s_,length p_) of p_)
487         if cycle p_ & cycle fi;
488     endfor;
489     p_
490 enddef;
491
492 % render a single segment - pulled out to make it easier to override
493 def tsu_render_segment(expr i,p,q) =
494     if do_alteration and obstackba.boalterate[i]:
495         default_nib:=fix_nib(obstackna.bosize[i]*tsu_brush_max
496                               *tsu_brush_shape,
497                               obstackna.bosize[i]*tsu_brush_max
498                               *tsu_brush_shape,
499                               0);
500     else:
501         default_nib:=fix_nib(obstackna.bosize[i]*tsu_brush_max,
502                               obstackna.bosize[i]*tsu_brush_max

```

```

503             *tsu_brush_shape,
504             tsu_brush_angle);
505 fi;
506 path mytip[],glyph;
507 for l=0 step 1 until length(p):
508     mytip[l]:=tsu_brush_tip(l,p,q,obstackna.bosize[i],s<1,
509         t>(length obstackp[i])-1,obstackba.boalternate[i]);
510 endfor;
511 pen_stroke(for ell=0 step 1 until length(p):
512     if sharp_corners and known obstacknaa.botip[i][ltime[ell]]:
513         tip(obstacknaa.botip[i][ltime[ell]])(ell)
514     else:
515         tsu_brush_opt(mytip[ell])(ell)
516     fi
517 endfor)(p if abs((point infinity of p)-(point 0 of p))<1:
518     -(point 0.001 of p)
519 fi)(glyph);
520 glstk[ngls]:=regenerate(glyph);
521 ngls:=ngls+1;
522 for l=0 step 1 until length(p):
523     si:=floor (ltime[l]+0.5);
524     if (abs(ltime[l]-si)<0.05) and known obstacknaa.boserif[i][si]:
525         tsu_serif.choose(obstacknaa.boserif[i][si],
526             point l of p,direction l of p,l,
527             obstackna.bosize[i],tsu_brush_tip_size(l,q));
528         write ("SERIF "&(decimal obstacknaa.boserif[i][si])&" "&
529             (decimal xpart point l of p)&""&
530             (decimal ypart point l of p)) to "proof.prf";
531     fi;
532 endfor;
533 enddef;
534
535 def tsu_render_in_circle(expr fitcircle) =
536 %
537 % find and apply rescaling xform
538 %
539 transform tsu_rescaling_xf;
540 tsu_rescaling_xf:=tsu_rescale_xform;
541 for i=1 upto sp-1:
542     if known obstackp[i]:
543         obstackp[i]:=obstackp[i] transformed tsu_rescaling_xf;
544     fi;
545     if known obstackt[i]:
546         obstackt[i]:=obstackt[i] transformed tsu_rescaling_xf;
547     fi;
548 endfor;
549 %
550 % main render

```

```

551 %
552 for i=1 upto sp-1: if obstacktype[i]=othook:
553     if obstackn[i]=hsmain_render:
554         scantokens obstacks[i];
555     fi;
556 fi; endfor;
557 begingroup
558     numeric i,j,k,l,s,t,si,ngls,flatl;
559     path bqi,p,q,glstk[];
560     ngls:=0;
561     flatl:=1;
562     for i=1 upto sp-1: if obstacktype[i]=otstroke:
563         if unknown obstackba.boalternate[i]:
564             obstackba.boalternate[i]:=false;
565         fi;
566 % message "suffix " & str i;
567         bqi:=obstackq[i] transformed tsu_brush_xf;
568         s:=0;
569         for j=0 step 1 until (length obstackp[i])-1:
570             k:=j+1;
571 % message " j=" & decimal j & " thr " & decimal (xpart point j of bqi)
572 % & "/" & decimal (xpart point k of bqi);
573             if ((xpart point j of bqi)<1)
574                 and ((xpart point k of bqi)>=1):
575 % message " START";
576                 if (xpart point k of bqi)=1:
577                     s:=k;
578                 else:
579                     s:=j+(xpart ((subpath (j,k) of bqi)
580                         intersectiontimes ((1,-infinity)
581                             -(1,infinity))));
582                 fi;
583             fi;
584             if (((xpart point j of bqi)>=1) and ((xpart point k of bqi)<1))
585                 or (k=length obstackp[i]):
586 % message " END";
587                 if (xpart point k of bqi)>=1:
588                     t:=k;
589                 else:
590                     t:=j+(xpart ((subpath (j,k) of bqi)
591                         intersectiontimes ((1,-infinity)
592                             -(1,infinity))));
593                 fi;
594                 if ((t-s)>0.02) and (obstackna.bosize[i]>0):
595                     boolean is_cycle;
596                     is_cycle:=((point s of obstackp[i])=(point t of obstackp[i]));
597                     p:=subpath (s,t) of obstackp[i];
598                     q:=subpath (s,t) of bqi;

```

```

599     numeric ltime[];
600     ltime[0]:=s;
601     for l=1 step 1 until (length p)-1:
602         ltime[l]:=floor (s+l);
603     endfor;
604     ltime[length p]:=t;
605     l:=0;
606     forever:
607         exitif l=length p;
608         begingroup
609             save x,y;
610             numeric x[],y[];
611             z10=segment_inflections(subpath (l,l+1) of p);
612             if known x10:
613                 p:=really_insert_nodes(p)(l+x10);
614                 q:=really_insert_nodes(q)(l+x10);
615                 for ll=length p step -1 until l+1:
616                     ltime[ll]:=ltime[ll-1];
617                 endfor;
618                 ltime[l+1]:=x10[ltime[l],ltime[l+2]];
619             else:
620                 z0=(point l of p)/100;
621                 z1=(postcontrol l of p)/100;
622                 z2=(precontrol (l+1) of p)/100;
623                 z3=(point (l+1) of p)/100;
624                 if if abs(z2-z1)>0.1: ((z1-z0) dotprod (z3-z2))
625                     /((z2-z1) dotprod (z2-z1))
626                     else: 1 fi<0.5:
627                     p:=really_insert_nodes(p)(l+0.5);
628                     q:=really_insert_nodes(q)(l+0.5);
629                     for ll=length p step -1 until l+1:
630                         ltime[ll]:=ltime[ll-1];
631                     endfor;
632                     ltime[l+1]:=0.5[ltime[l],ltime[l+2]];
633                 else:
634                     l:=l+1;
635                 fi;
636             fi;
637         endgroup;
638     endfor;
639     write ("SEGMENT "&(decimal flati)&" "&(decimal s)&" "&(decimal t))
640         to "proof.prf";
641     for lcbj=0 upto length p:
642         write ("POINT "&(decimal flati)&" "&(decimal ltime[lcbj])&" "
643             &(decimal xpart point lcbj of p)&" "
644             &(decimal ypart point lcbj of p)) to "proof.prf";
645     endfor;
646     flatl:=flatl+1;

```

```

647         tsu_render_segment(i,p,q);
648     fi;
649     fi;
650 endfor;
651 elseif obstacktype[i]=otlcblob;
652     glstk[nxls]:=regenerate(obstackp[i]);
653     nxls:=nxls+1;
654 fi; endfor;
655 %
656 % handle bounding circle
657 %
658 if xxpart fitcircle>0:
659     begingroup
660         save d,tmppt,pind,xpt,pts,pcnt,tmpxf;
661         pair pts[];
662         transform d;
663         pcnt:=0;
664         for j=0 upto nxls-1:
665             for i=0 step 0.1 until length glstk[j]:
666                 pts[pcnt]:=point i of glstk[j];
667                 pcnt:=pcnt+1;
668             endfor
669         endfor;
670         save lowpt; numeric lowpt;
671         lowpt:=0;
672         for i=0 upto pcnt-2:
673             for j=i+1 upto pcnt-1:
674                 if (i>=lowpt) and (j>=lowpt) and (abs(pts[i]-pts[j])<2):
675                     swap_pts(j,lowpt);
676                     lowpt:=lowpt+1;
677                 fi;
678             endfor;
679         endfor;
680         d:=bcircle.internal(lowpt,pcnt,pcnt);
681         transform tmpxf;
682         tmpxf=identity shifted (((0,0) transformed fitcircle)-
683                                 ((0,0) transformed d));
684         for j=0 upto nxls-1:
685             glstk[j]:=glsk[j] transformed tmpxf;
686         endfor;
687     endgroup
688 fi;
689 %
690 % finally render it all
691 %
692 for i=0 upto nxls-1:
693     dangerousFill glstk[i];
694 endfor;

```

```

695 %
696 % write misc. proof file stuff
697 %
698 blobcount:=0;
699 boxcount:=0;
700 for i=1 upto sp-1:
701   if obstacktype[i]=otlcblob:
702     begingroup
703       save spt,n;
704       pair spt;
705       spt:=(0,0);
706       n:=0;
707       for j=1 upto length obstackp[i]:
708         n:=n+1;
709         spt:=spt+(point j of obstackp[i]);
710       endfor;
711       spt:=spt/n;
712       blobcount:=blobcount+1;
713       write ("BLOBCENTRE "&(decimal blobcount)&" "
714         &(decimal xpart spt)&" "&(decimal ypart spt)) to "proof.prf";
715     endgroup;
716   elseif obstacktype[i]=otpbbox:
717     boxcount:=boxcount+1;
718     write ("PBOX "&
719       (decimal boxcount)&" "&
720       (decimal xpart ((0,0) transformed obstackt[i]))&" "&
721       (decimal ypart ((0,0) transformed obstackt[i]))&" "&
722       (decimal xpart ((1,0) transformed obstackt[i]))&" "&
723       (decimal ypart ((1,0) transformed obstackt[i]))&" "&
724       (decimal xpart ((1,1) transformed obstackt[i]))&" "&
725       (decimal ypart ((1,1) transformed obstackt[i]))&" "&
726       (decimal xpart ((0,1) transformed obstackt[i]))&" "&
727       (decimal ypart ((0,1) transformed obstackt[i]))&" "&
728       obstacks[i]&"") to "proof.prf";
729   if known obstackba.botoexpand[i]:
730     if obstackba.botoexpand[i]:
731       errmessage "Unexpanded PBOX: " & obstacks[i];
732     fi;
733   fi;
734   elseif obstacktype[i]=otanchor:
735     begingroup
736       save topanchor;
737       numeric topanchor;
738       topanchor:=i;
739       for j:=sp-1 downto i+1:
740         if obstacktype[j]=otanchor:
741           if obstackn[j]=obstackn[i]:
742             topanchor:=j;

```

```

743         fi;
744     fi;
745     exitif topanchor<>i;
746 endfor;
747 if topanchor=i;
748     write ("ANCHOR "&
749         (decimal obstackn[i]))&" "&
750         (decimal xpart ((-35,0) transformed obstackt[i]))&" "&
751         (decimal ypart ((-35,0) transformed obstackt[i]))&" "&
752         (decimal xpart ((35,0) transformed obstackt[i]))&" "&
753         (decimal ypart ((35,0) transformed obstackt[i]))&" "&
754         (decimal xpart ((0,35) transformed obstackt[i]))&" "&
755         (decimal ypart ((0,35) transformed obstackt[i]))&" "&
756         (decimal xpart ((0,35) transformed obstackt[i]))&" "&
757         (decimal ypart ((0,35) transformed obstackt[i]))
758         to "proof.prf";
759     fi;
760 endgroup;
761 fi;
762 endfor;
763 endgroup;
764 enddef;
765
766 % the usual case - just render it without fitting into a circle
767 def tsu_render =
768     tsu_render_in_circle(identity scaled -1);
769 enddef;
770
771 transform tsu_xf.smallkana;
772
773 tsu_xf.smallkana = identity shifted (-500,0) scaled 5.5/8 shifted (500,0);
774
775 def tsu_xform(expr xform)(text curves) =
776     begingroup
777         save txfsp,zc,zs;
778         txfsp:=sp;
779         curves;
780         numeric zs;
781         zs:=abs(((0,0) transformed xform)
782             -((1,0) transformed xform))
783             *abs(((0,0) transformed xform)
784             -((0,1) transformed xform));
785         size__scale:=zs*.16667;
786         for i=txfsp upto sp-1:
787             if known obstackp[i]:
788                 if known obstackba.bokeepshape[i]:
789                     if obstackba.bokeepshape[i]:
790                         pair zc;

```

```

791         zc:=0.5[(llcorner obstackp[i],urcorner obstackp[i]);
792         obstackp[i]:=obstackp[i] shifted (-zc) scaled (yyvspart xform)
793         shifted (zc transformed xform);
794     else:
795         obstackp[i]:=obstackp[i] transformed xform;
796     fi;
797 else:
798     obstackp[i]:=obstackp[i] transformed xform;
799     fi;
800 fi;
801 if known obstackna.bosize[i]:
802     obstackna.bosize[i]:=obstackna.bosize[i]*size__scale;
803 fi;
804 if known obstackt[i]:
805     if (xxpart obstackt[i]=1) and (yyvspart obstackt[i]=1)
806     and (xypart obstackt[i]=0) and (yxpart obstackt[i]=0):
807         obstackt[i]:=obstackt[i]
808         shifted (((0,0) transformed obstackt[i] transformed xform)-
809         ((0,0) transformed obstackt[i]));
810     else:
811         obstackt[i]:=obstackt[i] transformed xform;
812     fi;
813 fi;
814 endfor;
815 endgroup;
816 enddef;
817
818
819
820 def anc_upper = 1 enddef;
821 def anc_grave = 2 enddef;
822 def anc_acute = 3 enddef;
823 def anc_wide = 4 enddef;
824 def anc_tilde = 5 enddef;
825 def anc_ring = 6 enddef;
826 def anc_caron_comma = 7 enddef;
827 def anc_dakuten = 8 enddef;
828 def anc_lower = 9 enddef;
829 def anc_lower_connect = 10 enddef;
830 def anc_centre = 11 enddef;
831 def anc_iching_line(expr lnum) = (11+lnum) enddef;
832
833 transform accent_default[];
834 boolean accent_has_default[];
835 max_accent_seen:=0;
836
837 def tsu_default_anchor(expr aindex,avalue) =
838     if numeric avalue:

```

```

839 write ("DEFAULTANCHOR "&(decimal aindex)&" FALSE") to "proof.prf";
840 accent_has_default[aindex]:=false;
841 elseif transform avalue:
842   write ("DEFAULTANCHOR "&
843     (decimal aindex)&" "&
844     (decimal xpart ((0,0) transformed avalue
845       transformed tsu_rescale_xform))&" "&
846     (decimal ypart ((0,0) transformed avalue
847       transformed tsu_rescale_xform)))
848     to "proof.prf";
849   accent_has_default[aindex]:=true;
850 elseif pair avalue:
851   write ("DEFAULTANCHOR "&
852     (decimal aindex)&" "&
853     (decimal xpart (avalue transformed tsu_rescale_xform))&" "&
854     (decimal ypart (avalue transformed tsu_rescale_xform))
855     to "proof.prf";
856   accent_has_default[aindex]:=true;
857 fi;
858 if aindex>max_accent_seen:
859   max_accent_seen:=aindex;
860 fi;
861 enddef;
862
863
864
865 % figure out size of brush
866 vardef calc_mbrush_size =
867   numeric mbrush_width,mbrush_height,alternate_adjust;
868   (mbrush_width,mbrush_height)=urcorner (
869     fullcircle xscaled (tsu_brush_max*100)
870     yscaled (tsu_brush_max*tsu_brush_shape*100)
871     rotated tsu_brush_angle
872   );
873   alternate_adjust:=abs(mbrush_height-mbrush_width);
874   if tsu_brush_max>0.75:
875     serif_size:=2;
876   else:
877     serif_size:=2*((tsu_brush_max/0.75)**(1/3));
878   fi;
879   mincho_blob_size:=sqrt(tsu_brush_max/0.75);
880 enddef;
881
882 calc_mbrush_size;

```

## fntbase.mp

1 %  
2 % Tsukurimashou "font base" macros  
3 %  
4 % THIS FILE IS PUBLIC DOMAIN NOTWITHSTANDING THE COPYRIGHT ON THE  
5 % OVERALL TSUKURIMASHOU PACKAGE  
6 %  
7 % This file is based on the files "fontbase.mp" and "plain\_ex.mp" from the  
8 % METATYPE1 package version 0.55. Those files contain no copyright-related  
9 % notices of their own, but the README for METATYPE1 version 0.55 contains  
10 % the following notices (in English and Polish; the slashes are verbatim  
11 % from the original and presumably are some convention for expressing  
12 % non-ASCII Polish letters in the ASCII file):  
13 %  
14 % This is METATYPE1 package – a tool for creating Type 1 fonts using  
15 % METAPOST. The package belongs to public domain (no copyrights,  
16 % copyleft, copyups, copydowns, etc.).  
17 % Version: 0.55 (16.09.2009; a tentative version, released along with  
18 % the sources of the Latin Modern fonts ver. 2.003)  
19 % Author: JNS team <JNSteam@gust.org.pl>  
20 %  
21 % To jest pakiet METATYPE1 – narz/edzie do tworzenia font/ow Type 1  
22 % za pomoc/a systemu METAPOST. Pakiet stanowi dobro wsp/olne  
23 % (/zadnych copyright/ow, copyleft/ow, copyup/ow, copydown/ow, etc.).  
24 % Wersja: 0.55 (16.09.2009 – wersja opublikowana wraz z wersj/a  
25 % /xr/od/low/a 2.003 pakietu font/ow Latin Modern)  
26 % Autorstwo: JNS team <JNSteam@gust.org.pl>  
27 %  
28 % Although I assert my general right to claim copyright on work of my own  
29 % that draws from public domain source materials, I nonetheless am releasing  
30 % this file to the public domain in an effort to maintain the spirit of the  
31 % JNS team's release above.  
32 %  
33 % This program is distributed in the hope that it will be useful,  
34 % but WITHOUT ANY WARRANTY; without even the implied warranty of  
35 % MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  
36 %  
37 % Matthew Skala  
38 % mskala@ansuz.sooke.bc.ca  
39 %  
40  
41  
42

## General Library Functions

43 % GENERAL LIBRARY FUNCTIONS  
44

```

45 % inclusion lock written explicitly so as not to depend on preintro.mp
46 if known_already_included.fntbase:
47   endinput;
48 fi;
49 boolean already_included.fntbase;
50 already_included.fntbase:=true;
51
52 % gobble a text argument
53 def killtext text t = enddef; % absent from older versions of plain.mp
54
55 % Knuthian tradition unit definitions
56 mm#=2.84528; pt#=1; dd#=1.07001; bp#=1.00375; cm#=28.45276; pc#=12;
57 cc#=12.84010; in#=72.27;
58
59 % numeric functions
60 vardef tand primary a = sind(a)/cosd(a) enddef;
61 vardef cotd primary a = cosd(a)/sind(a) enddef;
62 vardef signum primary x = if x>0: 1 elseif x<0: -1 else: 0 fi enddef;
63 primarydef w dotnorm z =
64   begingroup
65     save w_, z_, lw_, lz_; pair w_, z_;
66     lw_=abs(w); w_:=w if lw_>0: /lw_ fi;
67     lz_=abs(z); z_:=z if lz_>0: /lz_ fi;
68     (xpart w_ * xpart z_ + ypart w_ * ypart z_)
69   endgroup
70 enddef;
71
72 % expand "decimal" to cover some other data types
73 let ori_decimal=decimal;
74 def decimal primary n =
75   (
76     if path n:
77       for i_=0 upto length(n)-1: if i_>0: & " " & fi
78       decimal(point i_ of n) & " " & decimal(postcontrol i_ of n) & " " &
79       decimal(precontrol i_+1 of n) & " " & decimal(point i_+1 of n)
80     endfor
81     elseif color n: ori_decimal(redpart(n)) & " " &
82     ori_decimal(greenpart(n)) & " " & ori_decimal(bluepart(n))
83     elseif pair n: ori_decimal(xpart(n)) & " " & ori_decimal(ypart(n))
84     else: ori_decimal(n) fi
85   )
86 enddef;
87
88 % The definition of lpostdir and lpredir given below is
89 % based on the following observation, being the consequence
90 % of l'Hôpital's rule: consider a Bézier segment
91 % la .. controls b and c .. dl; normally, the vector  $\vec{ab}$ 
92 % determines the "post" direction at node  $a$ ; if  $b$ 

```

```

93 % coincides with $a$, then the vector $\vec{ac}$ determines
94 % the direction; if also $c$ coincides with $a$,
95 % then the last resort is the vector $\vec{ad}$; if even $d$
96 % coincides with $a$, the Bézier segment is degenerated,
97 % and can be removed (a similar argumentation can be provided
98 % for the “pre” direction at node $d$).
99
100 % Previous, insufficiently robust definitions:
101 % \vardef predir expr t of p = (point t of p)-(precontrol t of p) enddef;|
102 % \vardef postdir expr t of p = (postcontrol t of p)-(point t of p) enddef;|
103 % \vardef udir expr t of p = unitvector(direction t of p) enddef;|
104
105 % New, more general definitions:
106 \vardef gendir expr t of p =
107   predir t of p + postdir t of p % |direction|-compatible definition
108 enddef;
109 \vardef predir expr t of p =
110   save a_,b_,c_,d_,s_,t_; pair a_,b_,c_,d_; path s_; t_:=t;
111   if not cycle p: if t<0: t_:=0; elseif t>length(p): t_:=length(p); fi fi
112   s_=subpath (ceiling t_-1,t_) of p;
113   a_=point 0 of s_;
114   b_=postcontrol 0 of s_; % |b_<>postcontrol t-1 of p| for |t=0|
115   c_=precontrol 1 of s_;
116   d_=point 1 of s_;
117   if d_<>c_: d_-c_
118   elseif d_<>b_: d_-b_
119   elseif d_<>a_: d_-a_
120   else: (0,0)
121   fi
122 enddef;
123
124 \vardef postdir expr t of p =
125   save a_,b_,c_,d_,s_,t_; pair a_,b_,c_,d_; path s_; t_:=t;
126   if not cycle p: if t<0: t_:=0; elseif t>length(p): t_:=length(p); fi fi
127   s_=subpath (t_,floor t_+1) of p;
128   a_=point 0 of s_;
129   b_=postcontrol 0 of s_;
130   c_=precontrol 1 of s_; % |c_<>precontrol t+1 of p| for |t=length p|
131   d_=point 1 of s_;
132   if a_<>b_: b_-a_
133   elseif a_<>c_: c_-a_
134   elseif a_<>d_: d_-a_
135   else: (0,0)
136   fi
137 enddef;
138
139 % Definitions related to “pre-” and “post-”
140 \vardef udir expr t of p = unitvector(gendir t of p) enddef;

```

```

141 vardef upredir expr t of p = unitvector(predir t of p) enddef;
142 vardef upostdir expr t of p = unitvector(postdir t of p) enddef;
143 vardef pos_subpath expr z of p =
144   if not cycle p: subpath z of p else:
145     if xpart(z)<=ypart(z): subpath z of p
146     else: subpath (xpart(z),ypart(z)+length(p)) of p fi
147   fi
148 enddef;
149
150 vardef posttension expr t of p = % "The \MF{}book", ex. 14.15
151   save q_; path q_;
152   q_:=point t of p {direction t of p} .. {direction t+1 of p} point t+1 of p;
153   length(postcontrol 0 of q_ - point 0 of q_)/
154   length(postcontrol t of p - point t of p)% doesn't work for "straight lines"
155 enddef;
156 vardef pretension expr t of p = % ditto
157   save q_; path q_;
158   q_:=point t-1 of p {direction t-1 of p} .. {direction t of p} point t of p;
159   length(precontrol 1 of q_ - point 1 of q_)/
160   length(precontrol t of p - point t of p)% doesn't work for "straight lines"
161 enddef;
162
163 % The two macros below, lpath_eq and linside macros, might have been
164 % primitives. The macro lpath_eq is obvious; linside returns true
165 % if the bounding box of l is inside the bounding box of b, which
166 % may be misleading; think, for example of:
167 % lfullcircle inside unitsquare shifted (-1/2,-1/2) scaled .9 rotated 45.
168 % For most curves occurring in fonts, however, one can safely infer
169 % that if l inside b holds, then l is inside b.
170 vardef path_eq(expr a,b)=
171   save i_,l_,r_; boolean r_;
172   r_:=((length(a)=length(b)) and (cycle a= cycle b));
173   if r_:
174     i_:=0; l_:=length(a) if cycle a: -1 fi;
175     forever:
176       r_:=((point i_ of a = point i_ of b); exitif not r_);
177       r_:=((precontrol i_ of a = precontrol i_ of b); exitif not r_);
178       r_:=((postcontrol i_ of a = postcontrol i_ of b); exitif not r_);
179       exitif incr i_>l_;
180     endfor fi
181   r_
182 enddef;
183
184 tertiarydef a inside b =
185   if path a: % land path b
186     (xpart llcorner b < xpart llcorner a) and
187     (xpart urcorner b > xpart urcorner a) and
188     (ypart llcorner b < ypart llcorner a) and

```

```

189 (ypart urcorner b > ypart urcorner a)
190 else: % lnumeric a and pair bl
191 (a>=xpart b) and (a<=ypart b)
192 fi
193 endif;
194
195 % The macro l&&l is to be used instead of the l&l operator if the respective
196 % ends of paths coincide only approximately; using l..l instead would add
197 % unwanted tiny B\ezier segments. The macro is somewhat "left-handed,"
198 % i.e., it does not consider the expression that follow the macro, therefore,
199 % it can be used before the 'lcyclel' command; if the argument lpl of the
200 % macro lamp_amp_l is a lpairl, it is just ignored which may be
201 % considered hardly intuitive.
202 def && = amp_amp_ whatever endif;
203 tertiarydef p amp_amp_ q =
204 if not pair p:
205 (subpath(0,length(p)-1) of p) .. controls (postcontrol length(p)-1 of p)
206 and (precontrol length(p) of p) ..
207 fi
208 endif;
209
210 vardef extrapolate expr t of b = % ltl pair, lbl B\ezier segment
211 clearxy;
212 Casteljau(xpart(t)) = point 0 of b;
213 Casteljau(1/3[xpart(t),ypart(t)]) = point 1/3 of b;
214 Casteljau(2/3[xpart(t),ypart(t)]) = point 2/3 of b;
215 Casteljau(ypart(t)) = point 1 of b;
216 z0 .. controls z1 and z2 .. z3
217 endif;
218
219 def Casteljau(expr t) =
220 t[t[z0,z1], t[z1,z2] ], t[t[z1,z2], t[z2,z3] ] ]
221 endif;
222
223 vardef elongation_to_times(expr ea,eb) =
224 % negative parameter values are admissible; they are meant for lpen_strokel
225 (if ea<0: - fi 1/(abs(ea)+1), eb/(abs(eb)+1))
226 endif;
227
228 % A numerical function 'lpoint_line_distl' takes as a parameter
229 % three lpairl expressions and returns a (signed) value of the distance
230 % of the first parameter from the line defined by the other two.
231 % It is referred to in the 'lis_linel' function.
232
233 vardef point_line_dist(expr a,b,c) =
234 clearxy; save d_; d_:=sqrt(length(b-c));
235 z0=a/d_; z1=b/d_; z2=c/d_;
236 (x2-x1)*(y1-y0)-(x1-x0)*(y2-y1)

```

```

237 endif;
238
239 % The idea of calculation of a turning angle
240 % between two vectors, employed in the definition of the function
241 % 'turn_ang,' is based on the following observation:
242 vardef turn_ang(expr za,zb) =
243   if (abs(za)>=1/1000) and (abs(zb)>=1/1000): % lepl may be not enough
244     angle(unitvector(za) zscaled (unitvector(zb) reflectedabout (origin,right)))
245   else: whatever fi
246 endif;
247
248 % A Boolean function 'lis_line1' checks whether a given B\ezier segment
249 % is a straight line. For large segments (fonts) it makes sense to specify
250 % a numerical parameter lis_line_off>=0; it defines a maximal acceptable
251 % distance of the control points of a B\ezier arc from its secant
252 % (which corresponds to the distance between the arc and the secant
253 % circa 13/4is_line_off1 for a symmetric, inflexionless arcs).
254 vardef is_line(expr B) =
255   save r_; boolean r_;
256   if known is_line_off:
257     save a_;
258     a_=length((point 1 of B)-(point 0 of B));
259     r_=(a_+arclength(B))<=(a_/infinity);
260     if r_:
261       r_=(is_line_off>=abs(point_line_dist(
262         postcontrol 0 of B, point 0 of B, point 1 of B))) and
263         (is_line_off>=abs(point_line_dist(
264         precontrol 1 of B, point 0 of B, point 1 of B)));
265     fi
266   else: % backward compatibility
267     save a_,b_,c_,d_;
268     a_=length((point 1 of B)-(point 0 of B));
269     b_=length((postcontrol 0 of B)-(point 0 of B));
270     c_=length((precontrol 1 of B)-(postcontrol 0 of B));
271     d_=length((point 1 of B)-(precontrol 1 of B));
272     r_=(a_+b_+c_+d_ <= a_/infinity);
273   fi
274   r_
275 endif;
276
277 % Abbreviations for a few simple yet useful phrases
278 def xyscaled primary p = xscaled xpart(p) yscaled ypart(p) endif;
279 def yxscaled primary p = yscaled xpart(p) xscaled ypart(p) endif;
280
281 % The macro linsert_nodes1 inserts additional nodes at given non-integer
282 % non-repeating times ltl into a given path lpl.
283 % The code would be a bit longer without 'larclength1' and 'larctimel.'
284 % The macro can be useful in some cases in the context of finding

```

```

285 % the envelopes of pen-stroked paths (avoiding inflection
286 % points—see below).
287 vardef insert_nodes(expr p)(text t) =
288   save j_, p_, s_, t_; path p_; p_:=p;
289   t_:=0;
290   for i_:=t:
291     if round(i_)<>i_: % ignore integer times
292       t_[incr t_]=arclength(subpath(0,i_ mod length(p_)) of p_);
293     fi
294   endfor
295   for i_:=1 upto t_:
296     s_:=arctime t_[i_] of p_;
297     if abs(round(s_)-s_)>eps: % ignore repeating times; is lepl OK?
298       p_:=subpath (0, s_) of p_ && (subpath (s_,length p_) of p_)
299       if cycle p_: & cycle fi;
300     fi
301   endfor;
302   p_
303 enddef;
304
305 % get rid of degeneracies
306 def regenerate(expr p) =
307   if (xpart urcorner p-xpart ulcorner p<5)
308     and (ypart ulcorner p-ypart llcorner p<5):
309     p
310   else:
311     begingroup
312       save q;
313       path q;
314       for t=1 step 1 until length p:
315         if abs((point t of p)-(point (t-1) of p))>3:
316           if unknown q:
317             q:=subpath (t-1,t) of p;
318           elseif length(q)=1:
319             q:=(point 0 of q)..
320               controls (postcontrol 0 of q) and (precontrol 1 of q)..
321               (0.5[point 1 of q,point t-1 of p])..
322               controls (postcontrol t-1 of p) and (precontrol t of p)..
323               (point t of p);
324           else:
325             q:=(subpath (0,length(q)-1) of q)..
326               controls (postcontrol length(q)-1 of q)
327               and (precontrol length(q) of q)..
328               (0.5[point length(q) of q,point t-1 of p])..
329               controls (postcontrol t-1 of p) and (precontrol t of p)..
330               (point t of p);
331         fi;
332       fi;

```

```

333     endfor;
334     if cycle p:
335         q:=subpath (0,length(q)-1) of q..
336         controls (postcontrol length(q)-1 of q)
337             and (precontrol length(q) of q)..
338         cycle;
339     fi;
340     q
341 endgroup
342 fi
343 enddef;
344
345 % like Fill, but doesn't complain about turning number
346 def dangerousFill text glist =
347 begingroup
348   save h_; path h_;
349   for g_:=glist:
350     h_:=g_ start.default; % JMN's suggestion
351     if glyph_usage div store = 1: % storing
352       glyph_stored.glyph_name[incr glyph_stored.glyph_name.num]=h_;
353     fi
354     glyph_list[incr glyph_list.num]:=round_node_values(h_ italicized);
355     update_glyph_bb(glyph_list[glyph_list.num]);
356   endfor;
357 endgroup
358 enddef;
359
360
361

```

---

## Prefix And Suffix Handling

```

362 % PREFIX AND SUFFIX HANDLING
363
364 % A method, entangled a bit and not particularly robust, of testing whether
365 % a parameter is a {\it string\} expression or a {\it suffix}.
366 % (Remark: lis_suffix((a))| or lis_suffix(a+b)| returns ltrue|;
367 % lis_suffix(((a)))| causes \MP{ } to report an error).
368 vardef is_suffix(text suffix_or_not_suffix) =
369   save the_suffix_; string the_suffix_; is_suffix_ suffix_or_not_suffix;
370   the_suffix_<>""
371 enddef;
372 def is_suffix_ suffix $ = the_suffix_:= str $; killtext enddef;
373
374 % suffix munging
375
376 def store_prec_obj = store_prec_obj_ whatever enddef;
377 primarydef a store_prec_obj_ b = hide(def prec_obj = a enddef) enddef;

```

```

378
379 % primarydef a sub b =
380 % if path a: (pos_subpath b of a) elseif string a: (substring b of a) fi
381 % enddef;
382
383 def node = store_prec_obj node__ enddef;
384 vardef node_@# primary a =
385   if str @#="x": xpart(point a of prec_obj)
386   elseif str @#="y": ypart(point a of prec_obj)
387   elseif str @#="": point a of prec_obj
388   else:
389     errhelp "The operator 'node' works only with 'x', 'y' or an empty suffixes.";
390     errmessage "PX: improper usage of 'node'";
391   fi
392 enddef;
393
394 def first suffix $ =
395   if str $="at": % moves the first point of a path to a specified location
396     store_prec_obj prec_obj shifted -(point 0 of prec_obj) shifted
397   else: node$(0) fi
398 enddef;
399 def last suffix $ =
400   if str $="at": % moves the last point of a path to a specified location
401     store_prec_obj prec_obj shifted
402     -(point if cycle prec_obj: 0 else: infinity fi of prec_obj) shifted
403   else: node$(if cycle prec_obj: 0 else: infinity fi) fi
404 enddef;
405
406 % Neat macros excerpted from John D. Hobby's boxes.mp macro package
407
408 % Find the length of the prefix of string lsl for which lcondl is true for
409 % each character lcl of the prefix
410 vardef genericize_prefix(expr s)(text cond) =
411   save i_, c_; string c_;
412   i_ = 0;
413   forever:
414     c_ := substring (i_,i_+1) of s;
415     exitunless cond; exitif incr i_=length s;
416   endfor
417   i_
418 enddef;
419
420 % Take a string returned by the lstrl operator and return the same string
421 % with explicit numeric subscripts replaced by generic subscript symbols []:
422 vardef genericize(expr s) =
423   save res_, s_, l_; string res_, s_;
424   res_=""; % result so far
425   s_ =s; % left to process

```

```

426 forever: exitif s_="";
427 l_:=genericize_prefix(s_, (c_<>"[") and ((c_<"0") or (c_>"9")));
428 res_:=res_ & substring (0,l_) of s_;
429 s_:=substring (l_,infinity) of s_;
430 if s_<>"":
431   res_ := res_ & "[";
432   l_ :=if s_>="[": 1+genericize_prefix(s_, c_<>"]")
433   else: genericize_prefix(s_, (c_="") or ("0"<=c_) and (c_<="9")) fi;
434   s_:=substring(l_,infinity) of s_;
435 fi
436 endfor
437 res_
438 enddef;
439
440
441

```

---

## A Module That Finds An Envelope Of A Path Drawn With A Pen

**FNTB**

```

442 % A MODULE THAT FINDS AN ENVELOPE OF A PATH DRAWN WITH A PEN
443
444 % The following macros approximate the envelope of an elliptical or a razor
445 % pen. The exact solution is impossible—in general, the envelope is not
446 % a B\'ezier curve, therefore some heuristics is, in general, unavoidable.
447 % We assumed that the backbone of a figure is such that
448 % the envelope does not form loops at smoothly joined nodes. Moreover,
449 % all B\'ezier segments appearing in the process {\bf should not}
450 % contain inflection points (the reason for this limitation is the
451 % method of finding an approximation of a pen envelope). If the latter
452 % condition is not fulfilled, one may expect weird results (see the usage
453 % of the |...| operator in the code of lpen_stroke_edgel).
454
455 % We assume that slanting should not distort a pen. Therefore, if
456 % a glyph is to be slanted {\it after\/} expanding a stroke, which
457 % usually is the case, the envelope should be constructed with
458 % an {\it unslanted pen}. Macros lslant_strokel, lunslant_strokel,
459 % and lunslant_angle are devised to facilitate handling this
460 % situation. These macros refer to the variable lslant_stroke_val;
461 % it should be assigned a definite value prior to expanding stroke.
462 def slant_stroke =
463   if known slant_stroke_val: slanted slant_stroke_val fi
464 enddef;
465 def unslant_stroke =
466   if known slant_stroke_val: slanted -slant_stroke_val fi
467 enddef;
468 vardef unslant_angle(expr a) = angle(dir(a) unslant_stroke) enddef;
469

```

```

470 % Macro lfix_nibl returns a path. If ly_diaml parameter
471 % is 0, a "razor" pen (a segment) is returned, otherwise it is
472 % an approximation of an ellipse. We do our best to avoid unnecessary
473 % nodes, hence the approximation is somewhat complicated; another reason
474 % for the complication is that interpolation and affine transformations
475 % do not commute, therefore the appropriate nodes are found for
476 % the untransformed pen, and only then the pen is transformed.
477 % {\it Note\}: So far, there is no explicit relation between a built-in
478 % \MP{} pen mechanism and the lfix_nibl operation, in particular,
479 % lbeginfigl does not alter the setting of ldefault_nibl. Needs rethinking.
480
481 vardef fix_nib(expr x_diam, y_diam, rot_angle) =
482   if (x_diam<>0) and (y_diam<>0): fix_elliptic_nib(x_diam, y_diam, rot_angle)
483   elseif (x_diam<>0) and (y_diam=0): fix_razor_nib(x_diam, rot_angle)
484   elseif (x_diam=0) and (y_diam<>0): fix_razor_nib(y_diam, rot_angle+90)
485   else:
486     errhelp "I'll use the default pen, but I'd suggest to cancel the job.";
487     errmessage "PX: the null pen is not allowed";
488     default_nib
489   fi
490 enddef;
491
492 vardef fix_razor_nib(expr x_diam, rot_angle) =
493   ((-1/2x_diam,0)-(1/2x_diam,0)) rotated rot_angle unslant_stroke
494 enddef;
495
496 vardef fix_elliptic_nib(expr x_diam, y_diam, rot_angle) =
497   save p_; path p_;
498   % construct a temporary ellipse:
499   p_:=fullcircle
500   xscaled x_diam yscaled y_diam rotated rot_angle unslant_stroke;
501   % construct an elliptic pen path having
502   % 4 or, if necessary (heuristic), 6 nodes:
503   (for d=up unslant_stroke, left,
504     if (y_diam/x_diam<1/2) and (abs(rot_angle mod 90)>5):
505       left rotated rot_angle unslant_stroke,
506     fi
507     down unslant_stroke, right,
508     if (y_diam/x_diam<1/2) and (abs(rot_angle mod 90)>5):
509       right rotated rot_angle unslant_stroke
510     fi:
511     (point(directiontime d of p_) of fullcircle)
512     {direction (directiontime d of p_) of fullcircle}...
513   endfor cycle) xscaled x_diam yscaled y_diam rotated rot_angle unslant_stroke
514 enddef;
515
516 % Arcs of a pen shorter than lignore_nib_limitl will be joined together
517 % to form larger ones. Remember to adjust the parameter lignore_nib_limitl

```

```

518 % if the size of ldefault_nibl is significantly changed.
519 newinternal ignore_nib_limit; ignore_nib_limit:=1;
520
521 path default_nib;
522 default_nib:=fix_nib(50,50,0); % hundred times as large as a default plain pen
523
524 newinternal default_elongation, default_join, default_cap;
525 default_elongation:=1/2;
526 default_join:=1;
527 % 0 – tip, default elongation used
528 % 1 – pen join, default elongation ignored
529 % 2 – tip, default elongation ignored, elongation=0 used
530 default_cap:=1;
531 % 0 – cut 90 rel
532 % 1 – pen end
533
534 % ltangent_pointl, lpen_joinl, lpen_stroke_edge_l, and lpen_stroke_edgel
535 % are auxiliary macros, exploited by the main macro, i.e., lpen_strokel.
536 vardef tangent_point(expr d,nib) = % |d| – direction of pen movement
537   save a_;
538   point if cycle nib: (directiontime d of nib) else:
539     hide (a_:=turn_ang(d,(point 1 of nib)-(point 0 of nib)))
540     if abs(a_ mod 180)<.1: 1/2 % emergency
541       elseif a_<0: 0 else: 1 fi
542   fi of nib
543 enddef;
544
545 vardef pen_join(expr a,b,c,nib)=
546 % deleting superfluous nodes is based on the larclengthl operation
547 % which, obviously, is not preserved after slanting, but let's hope
548 % it does not matter (too much)
549   save t_, m_, m___, ta_, tb_, p_; path p_;
550   m_:=infinity; % will be the minimal length of lnibl's segment
551   for t_:=0 upto 1/2length(nib)-1:
552     m___:=arclength(subpath(t_,t_+1) of nib);
553     if m___<m_: m_:=m___; fi
554   endfor
555   if m_<ignore_nib_limit:
556     message "PX: the shortest nib segment < ignore_nib_limit (" &
557       decimal(m___) & " < " & decimal(ignore_nib_limit) & ")";
558   fi
559   p_:=nib shifted c;
560   if cycle nib:
561     ta_=directiontime a of p_; tb_=directiontime b of p_;
562     p_:=pos_subpath(ta_,tb_) of p_;
563     if arclength(p_)>ignore_nib_limit:
564       for i_:=0,0:
565         p_:=reverse p_; % short segments may appear at both ends

```

```

566   if length(p_)>1: % optimization
567       if arclength(subpath (0,1) of p_)<1/4ignore_nib_limit:
568           % cf. the comment concerning 1/4ignore_nib_limit! in
569           % lpen_stroke_edgel below
570       p_:=((point 0 of p_) .. controls (postcontrol 1 of p_) and
571           (precontrol 2 of p_) .. subpath (2,infinity) of p_);
572   fi
573   fi
574   endfor
575   else:
576       p_:=((point 0 of p_){a}...{b}(point length(p_) of p_);
577   fi
578   else: % razor nib
579       p_:=tangent_point(a,p_)-tangent_point(b,p_);
580   fi
581   p_
582 enddef;
583
584 % The finding of a pen envelope for a given B\ezier segment,
585 % defined by nodes la, lb, lc, and ld, begins with
586 % the placing the pen at the ends of the B\ezier segment
587 % (i.e., at the points la, ld) and finding the corresponding points
588 % la' and ld' where the pen outline is parrallel to the direction
589 % of the original path at these points. Then, the outline is constructed.
590 % For lpen_stroke_method=0l (default), the envelope segment is constructed
591 % by setting the beginning and final directions (optionally, the direction
592 % at a given node can be ignored); for lpen_stroke_method=1l or 2
593 % an alternative (more elaborate) procedure is involved which explicitly
594 % computes control nodes lb' and lc' of the resulting path basing on
595 % a heuristic assumption that
596 %  $||length(b'-a')/length(b-a)|| \approx$ 
597 %  $||length(c'-d')/length(c-d)|| \approx$ 
598 %  $||length(a'-d')/length(a-d)||$ . \break
599 % The default method never produce concave edges because the operator l...l
600 % is used always; the alternative methods employs the operator
601 % lforce_convex_edgel instead; for lpen_stroke_method=1l the convex edges
602 % are forced (i.e, inflexion points are being removed),
603 % for lpen_stroke_method=2l no forcing of convex edges takes place.
604 vardef extrapoline expr t of B = % the result may be not a single segment
605   save l_, t_;
606   (t_.a,t_.b)=t; % l0<=ta<tb<=1l!
607   l_=arclength(B)/(t_.b-t_.a); l_.a=l_*t_.a; l_.b=l_*(1-t_.b);
608   if t_.a>0: ((point 0 of B) - l_.a*(upostdir 0 of B))- fi
609   B
610   if t_.b<1: - ((point 1 of B) + l_.b*(upredir 1 of B)) fi
611 enddef;
612
613 vardef force_convex_edge(expr za, zb, zc, zd) =

```

```

614 save a_, b_, c_, d_, z_;
615 a_:=length(zd-za); b_:=length(zb-za); c_:=length(zc-zb); d_:=length(zd-zc);
616 if (-a_+b_+c_+d_ > a_/infinity):
617   if pen_stroke_method=2:
618     za .. controls zb and zc .. zd
619   else:
620     if (a_>0.01) and (b_>0.01) and (c_>0.01) and (d_>0.01): % no degeneration...
621       a_:=signum((za-zd) rotated -90 dotnorm (zb-za));
622       b_:=signum((zb-za) rotated -90 dotnorm (zc-zb));
623       c_:=signum((zc-zb) rotated -90 dotnorm (zd-zc));
624       d_:=signum((zd-zc) rotated -90 dotnorm (za-zd));
625       if ((a_<>b_) or (b_<>c_)) and (a_=d_):
626         numeric b_, c_; pair z_;
627         z_=b_[za,zb]=c_[zd,zc];
628         za .. controls
629           if b_<1: z_ else: zb fi and if c_<1: z_ else: zc fi
630         .. zd
631       else:
632         za .. controls zb and zc .. zd
633       fi
634     else:
635       za .. controls zb and zc .. zd
636     fi
637   fi
638 else:
639   za - zd
640 fi
641 enddef;
642
643 vardef pen_stroke_edge_(expr b,b_nib,e_nib) = % |b| - B\ezier segment
644 save pa_,pb_,qa_,qb_,ra_,rb_,sa_,sb_;
645 pair pa_,pb_,qa_,qb_,ra_,rb_,sa_,sb_;
646 pa_=point 0 of b; ra_=(postcontrol 0 of b)-pa_; sa_=postdir 0 of b;
647 pb_=point 1 of b; rb_=(precontrol 1 of b)-pb_; sb_=predir 1 of b;
648 qa_=pa_ + tangent_point(sa_, b_nib);
649 qb_=pb_ + tangent_point(sb_, e_nib);
650 if pen_stroke_method=0:
651   qa_ {sa_} ... {sb_} qb_
652 elseif (pen_stroke_method=1) or (pen_stroke_method=2):
653   save lp_,lq_; lp_=length(pb_-pa_); lq_=length(qb_-qa_);
654   if 2lp_<lq_: % heuresis - too close nodes
655     qa_ {sa_} ... {sb_} qb_
656   else:
657     force_convex_edge(qa_, qa_+lq_/lp_*ra_, qb_+lq_/lp_*rb_, qb_)
658   fi
659 else:
660   errhelp "Only the values 0,1 and 2 for 'pen_stroke_method' are admissible. " &
661   "Better stop now.";

```

```

662 errmessage "PX: unknown pen stroke method (" &
663   decimal(pen_stroke_method) & ")";
664 fi
665 enddef;
666
667 vardef pen_stroke_edge@#(expr p) =
668   save e_,l_,i_,i__; path e_[\];
669   l_:=length(p);
670   for i_:=0 upto l_-1:
671     e_[i_]:=pen_stroke_edge_(subpath (i_,i_+1) of p,
672       % |local_nib_@#(i_),local_nib_@#(i_+1)| % a nasty bug removed 20.08.2009
673       local_nib_@#(i_),local_nib_@#((i_+1) if cycle p: mod l_ fi));
674   endfor
675   for i_:=0 upto l_ if cycle p: -1 else: -2 fi:
676     i__:= (i_+1) mod l_;
677     save t_;
678     t_:=turn_ang(predir 1 of e_[i_], postdir 0 of e_[i__]);
679     if if known t_: abs(t_)>1 else: false fi:
680       save t_; (t_.a,t_.b)=e_[i_] intersectiontimes e_[i__];
681       if t_.a>0:
682         e_[i_]:=subpath (0,t_.a) of e_[i_];
683         e_[i__]:=subpath (t_.b,1) of e_[i__];
684       elseif known local_tip_@#(i__):
685         save tx_, ty_, b_, b__, ei_, ei__; path ei_, ei__, ei_[], ei__[];
686         (tx_,ty_)=local_tip_@#(i__);
687         ei_:=if is_line(e_[i_]):
688           (point 0 of e_[i_])--
689           (1/abs(tx_))[point 0 of e_[i_], point 1 of e_[i_] ]
690         elseif tx_<0: hide(b_:=1) extrapoline (0,abs(tx_)) of e_[i_]
691         else: extrapolate (0,abs(tx_)) of e_[i_] fi;
692         ei__:=if is_line(e_[i__]):
693           (1/(1-abs(ty_)))[point 1 of e_[i__], point 0 of e_[i__] ] --
694           point 1 of e_[i__]
695         elseif ty_<0: hide(b__:=1) extrapoline (abs(ty_),1) of e_[i__]
696         else: extrapolate (abs(ty_),1) of e_[i__] fi;
697       % clumsy HEURESIS (choosing an optimal intersection point, if there are
698       % more intersections):
699       save t_; (t_.a1,length(ei__)-t_.b1)=ei_ intersectiontimes reverse ei__;
700       if t_.a1>0:
701         ei_1:=if (known b_) and (t_.a1>1):
702           force_convex_edge(point 0 of e_[i_], postcontrol 0 of e_[i_],
703             precontrol 1 of e_[i_], point t_.a1 of ei_)
704         else: subpath (0,t_.a1) of ei_ fi;
705         ei__1:=if (known b__) and (t_.b1<1):
706           force_convex_edge(point t_.b1 of ei__, postcontrol 0 of e_[i__],
707             precontrol 1 of e_[i__], point 1 of e_[i__])
708         else: subpath (t_.b1,infinity) of ei__ fi;
709       (length(ei_)-t_.a2,t_.b2)=reverse ei_ intersectiontimes ei__;

```

```

710     if length((t_.a1,t_.b1)-(t_.a2,t_.b2))>eps:
711         ei_2:=if (known b_) and (t_.a2>1):
712             force_convex_edge(point 0 of e_[i_], postcontrol 0 of e_[i_],
713                 precontrol 1 of e_[i_], point t_.a2 of ei_)
714         else: subpath (0,t_.a2) of ei_ fi;
715         ei__2:=if (known b__) and (t_.b2<1):
716             force_convex_edge(point t_.b2 of ei__, postcontrol 0 of e_[i__],
717                 precontrol 1 of e_[i__], point 1 of e_[i__])
718         else: subpath (t_.b2,infinity) of ei__ fi;
719         if arclength(ei_1)+arclength(ei__1) > arclength(ei_2)+arclength(ei__2):
720             ei_1:=ei_2; ei__1:=ei__2;
721         fi
722     fi
723     e_[i_]:=ei_1; e_[i__]:=ei__1;
724 fi
725 fi
726 fi
727 endfor
728 for i_:=0 upto l_-1:
729     hide(i__:=(i_-1) mod l_)
730     if cycle p or (i_>0):
731         if length((point 1 of e_[i__])-(point 0 of e_[i_]))>1/4ignore_nib_limit:
732             % the constant 1/4ignore_nib_limit plays a similar role
733             % to that of the ISNAP_TO_NODE variable in pf2mt1.awk
734             (point 1 of e_[i__])
735             if known local_tip_@#(i_): – else:
736                 && pen_join(predir 1 of e_[i__],postdir 0 of e_[i_],point i_ of p,
737                     local_nib_@#(i_)) &&
738             fi
739         fi
740     fi
741     % reconstruct le_[i_] (possibly ignoring direction(s)):
742     (point 0 of e_[i_])
743     if is_line(e_[i_]):
744         % the using of |...| circumvents \MF{}/\MP{} instable behaviour:
745         % the operator |...| may cause that a control point and a node
746         % (nearly) coincide (note that this is feature, not a bug);
747         % thus, it is advisable for |pen_stroke_method=0|; supposedly,
748         % it is also adequate for |pen_stroke_method=1|:
749         –
750     else:
751         if pen_stroke_method=0:
752             if not ignore_dir_(i_): {postdir 0 of e_[i_]} fi ...
753             if not ignore_dir_(i_+1): {predir 1 of e_[i_]} fi
754         elseif (pen_stroke_method=1) or (pen_stroke_method=2):
755             .. controls (postcontrol 0 of e_[i_]) and (precontrol 1 of e_[i_]) ..
756         fi
757     fi

```

```

758 endfor
759 if cycle p: cycle else: (point 1 of e_[l_-1]) fi
760 enddef;
761 newinternal pen_stroke_method;
762
763 % Macro lpen_stroke performs an operation known as “expanding stroke”;
764 % we’ll call the result of the operation a “pen envelope” (for
765 % a given path). The macro has one optional parameter, loptsl (ltextl),
766 % and two obligatory ones: input path lpl (lexprl)
767 % and a lresultl (lsuffixl). A user has an access to subpaths of the
768 % envelope, namely: lresult.rl is the right edge of the envelope,
769 % lresult.ll—its left edge, lresult.bl—is a fragment of the pen outline
770 % joining left and right edge of the envelope at the beginning
771 % node of the path, lresult.el—is a similar fragment at the ending
772 % node of the path (see the picture below). If the path lpl
773 % is cyclic, then lresult.el and lresult.bl are undefined,
774 % otherwise the variable lresultl contains additionally the complete
775 % expanded stroke.
776
777 % For finding an envelope, a default path (ldefault_nibl, returned
778 % by lfix_nibl) is used except nodes for which the parameter loptsl
779 % sets another pen. Mastering the usage of the parameter loptsl allows
780 % a user to achieve nontrivial effects. The parameter loptsl is a list
781 % (space-separated or semicolon-separated) of the following
782 % operators: (1) lniibl, (2) lcutl, (3) ltipl, and (4) lignore_directionsl.
783
784 % Ad 1. The macro lniibl has two parameters:
785 % lniibl(pen)(list_of_nodes), where “pen” is a path returned by
786 % macro lfix_nibl, and “list_of_nodes” contains comma-separated numbers
787 % (times) of the nodes of the path lpl at which a given pen is to be
788 % used. If needed, the outline is complemented at corner nodes
789 % with a fragment of a pen path. Such a join corresponds to the setting
790 % llinejoin:=roundedl in \MP{. If the path lpl is non-cyclic,
791 % its ends are also complemented with appropriate fragments of a pen path
792 % (the setting llinecap:=roundedl). Such a method of joining is also applied
793 % by lpen_stroke to nodes not mentioned in the parameter loptsl.
794 % The result of the following statement
795 % \LINE{\descriptioncomments
796 % lpen_stroke(nib(default_nib xyscaled (1,2))(infinity))(p)(q)l
797 % \unskip}
798 % \descriptioncomments
799 % that changes the pen at the last node of the path,
800 % is shown in the following picture:
801 % \LINE{\epsfbox{\illusname.110}}
802
803 % Ad 2. The call of the macro lcutl has the form: lcutl(angle,
804 % pen)(list_of_nodes) or lcutl(pen, angle)(list_of_nodes),
805 % where “pen” and “list_of_nodes” are defined as

```

```

806 % previously. The pen parameter can be omitted which means using a default
807 % pen (ldefault_nib). The macro replaces a default pen with a special
808 % “razor” pen at specified nodes. More precisely, it is a projection of a
809 % given pen in the direction of the path |p| at a given node onto a
810 % straight line going through this node under the angle specified in the
811 % respective parameter of the macro. Uf\ /f\ /f\dots\ The angle of the straight
812 % line can be defined either absolutely (with respect to the axis |x|)
813 % or—by adding a prefix ‘lrel’—relatively to the direction of the path
814 % at a given node. From the point of view of a user, the result of the
815 % macro lcutl is “cutting” the expanded stroke with a straight
816 % line. This operation is particularly useful at the ends of a path and
817 % corresponds to the setting llinecap:=buttl in \MP{ }, except that in \MP{ }
818 % one cannot specify angles. The result of the statement
819 % \LINE{descriptioncomments
820 % lpen_stroke(cut(45)(0)|
821 % lcut(default_nib xyscaled (1,2), rel 90)(infinity))(p)(q)|
822 % \unskip}
823 % \descriptioncomments
824 % that cuts both ends and, moreover, changes a pen
825 % at the ending node is shown in the figure below
826 % (at the beginning node, the absolute angle of 45 degrees is specified,
827 % at the ending one—the relative angle of 90 degrees):
828
829 % Ad 3. The call of the macro ltipl has the form ltipl(pen,
830 % pre_elongate, post_elongate)(list_of_nodes), where “pen”
831 % and “list_of_nodes” have the same meaning as previously.
832 % In particular, a pen can be omitted. At corner nodes
833 % specified in the list of nodes, the consecutive elements of the outline
834 % are not joined with an appropriate subpath of a pen; instead, they
835 % are elongated (extrapolated) until they intersect. This process corresponds
836 % (roughly) to the \MP{ } setting llinejoin:=miteredl:
837
838 % The illustration above is the result of the following call
839 % of the macro lpen_stroke (the macro ltipl is invoked with default
840 % settings, only the number of a node is specified):
841 % lpen_stroke(tip()(3))(p)(q); draw q;|
842 % The optional parameters lpre_elongationl and lpost_elongationl define how
843 % far the consecutive edges (segments) should be elongated in order to make
844 % them intersect each other (the measure is the time). If one parameter is
845 % omitted, both will receive the same value; if both are omitted, a default
846 % value, l(0.5,0.5)l (it corresponds to elongation by circa 50%), will be
847 % used. The precise meaning of the pre- and post-elongation is defined as
848 % follows: for a given pre-edge le1l, post-edge le2l, pre-elongation lv1l
849 % and post-elongation lv2l, the paths
850 % lextrapolate (0, 1/(1+v1) of e1l and
851 % lextrapolate (v2/(1+v2), 1) of e2l are computed
852 % (i.e., for the default elongation: lextrapolate (0, 2/3) of s1l
853 % and lextrapolate (1/3, 1) of s2l, respectively).

```

```

854 % If elongated curves do not intersect, the terminal nodes
855 % of the consecutive segments are joined with a straight line. The latter
856 % property can be used to obtain a result corresponding to the \MP{ } setting
857 % llinejoin:=beveled: it suffices to apply a null elongation, i.e.,
858 % ltipl(0)(list_of_nodes). Changing the first (empty) parameter
859 % of the ltipl macro in the previous example would yield the following
860 % result:
861
862 % Ad 4. The macro lignore_directionsl has a different character. It is
863 % invoked with one parameter being a comma-separated list of nodes:
864 % lignore_directionsl(list_of_nodes). The numbers {\it must be\ } followed
865 % by suffixes lll or lrl. The macro causes that, at specified nodes,
866 % the direction of the outline is not forced to be parallel to the direction
867 % of the path lpl (which is the default); instead, the direction is
868 % calculated by \MP{ }. Suffixes determine whether the direction
869 % is not to be forced at the right (lrl) or the left (lll) edge (with
870 % respect to the direction of the path lpl). This heuristic
871 % trick can be used to improve the appearance of the outline
872 % if the "inner" part of the envelope has too tight arcs.
873 %% The examples of the usage of this macro can be found in the \MP{ } version
874 %% of D. E. Knuth's 'logo' font (letters 'P' and 'S').
875
876 vardef pen_stroke(text opts)(expr p)(suffix result) =
877   forsuffices $=,r,l,b,e:
878     if not path result$: scantokens("path " & genericize(str result$)); fi
879   endfor
880   save a_, a__, d_, i_, k_, n_, p_, z_, norm_, norml_, normr_, normlr_,
881     fix_opts_, ignore_dir_, ignore_dir__, local_nib_, local_nib__,
882     local_tip_, default_tip_, local_tip__, % internal
883     all, rel, last, nib, cut, tip, ignore_directions, current_node; % exported
884   numeric ignore_dir__[ ]; pair default_tip_, local_tip__[ ];
885   path local_nib__[ ];
886   pair a_, d_, z_[ ]; path p_;
887   %% xpart norm_ norml_ normr_ normlr_
888   vardef norm_ primary n =
889     if cycle p: n mod last else: if n<0: 0 elseif n>last: last else: n fi fi
890   enddef;
891   vardef norml_ primary n = -norm_ n -1 enddef;
892   vardef normr_ primary n = norm_ n +1 enddef;
893   vardef normlr_@# primary i= if str @#="l": -norm_(last-i)-1 else: i+1 fi enddef;
894   last=length(p);
895   vardef rel primary a =
896     angle((gendir current_node of p) slant_stroke)+a
897   enddef;
898   def all =
899     hide(% locally we use the prefix rather than postfix notation;
900       % a trick due to the lsuffixl parameter of the lallcont_l macro
901       vardef l primary n = (norml_ n,0) enddef;

```

```

902   vardef r primary n = (normr_ n,0) enddef) allcont_
903 enddef;
904 def allcont_ suffix $ =
905   $0 for i_:=1 upto last if cycle p: -1 fi: , $i_ endfor
906 enddef;
907 vardef fixopts_(suffix optname)(text nodes) text val =
908 %% intersectiontimes lcont_ rcont_
909   save l, r, lcont_, rcont_;
910   def l = lcont_ whatever enddef; primarydef a lcont_ b = (norml_ a,0) enddef;
911   def r = rcont_ whatever enddef; primarydef a rcont_ b = (normr_ a,0) enddef;
912   for n_:=nodes:
913     if numeric n_:
914       current_node:=norm_ n_;
915       optname[norml_ n_]:=optname[normr_ n_]
916     else:
917       current_node:=abs(xpart n_)-1; % the inverse of both lnorml_| and lnormr_|
918       optname[xpart(n_)]
919     fi :=val; % lval may depend on lcurrent_node
920   endfor
921 enddef;
922 def nib(text nib_)(text nodes) = % nib and node list
923   fixopts_(local_nib_)(nodes)
924   begingroup
925     p_:=default_nib; for k_:=nib_: p_:=k_; endfor \ p_
926   endgroup;
927 enddef;
928 def cut(text nib_and_ang)(text nodes) = % angle, nib and node list
929   fixopts_(local_nib_)(nodes)
930   begingroup
931     p_:=default_nib;
932     for k_:=nib_and_ang:
933       if numeric k_: a_:=dir(unslant_angle(k_)); else: p_:=k_; fi
934     endfor
935     d_:=gendir current_node of p;
936     z_1:=whatever*a_:=tangent_point(d_,p_)+whatever*d_;
937     z_2:=whatever*a_:=tangent_point(-d_,p_)+whatever*d_;
938     z_1-z_2
939   endgroup;
940 enddef;
941 def tip(text nib_and_lim)(text nodes)= % limit(s) and node list
942   i_:=0; for n_:=nib_and_lim: if numeric n_: i_[incr i_]:=n_; fi endfor
943   fixopts_(local_tip_)(nodes)
944   elongation_to_times(if i_:=0: default_elongation, default_elongation
945     elseif i_:=1: i_1, i_1 else: i_1, i_2 fi);
946   fixopts_(local_nib_)(nodes)
947   begingroup
948     p_:=default_nib; for k_:=nib_and_lim: if path k_: p_:=k_; fi endfor \ p_
949   endgroup;

```

```

950 endif;
951 def ignore_directions(text nodes) = % node list
952   fixopts_(ignore_dir__)(nodes) 1;
953 endif;
954 if default_cap=0:
955   if not cycle p: cut(rel 90)(0,last); fi
956 elseif default_cap=1: % do nothing
957 else:
958   errhelp "Admissible values are 0, 1; continue, I'll use the value 1!";
959   errmsg "PX: improper 'default_cap' value ("&decimal(default_cap)&")";
960 fi
961 opts;
962
963 if default_join=0:
964   default_tip_:=elongation_to_times(default_elongation, default_elongation);
965 elseif default_join=1: % no tip setting, do nothing
966 elseif default_join=2:
967   default_tip_:=1,0; % 1(1,0)=elongation_to_times(0,0)
968 else:
969   errhelp "Admissible values are 0, 1, 2; continue, I'll use the value 1!";
970   errmsg "PX: improper 'default_join' value ("&decimal(default_join)&")";
971 fi
972 vardef ignore_dir_@#(expr i) = known ignore_dir__[normlr_@# i] endif;
973 vardef local_tip_@#(expr i) = if known local_tip__[normlr_@# i]:
974   local_tip__[normlr_@# i] else: default_tip_ fi endif;
975 vardef local_nib_@#(expr i) = if known local_nib__[normlr_@# i]:
976   local_nib__[normlr_@# i] else: default_nib fi endif;
977 result.r:=pen_stroke_edge.r(p);
978 result.l:=pen_stroke_edge.l(reverse p);
979 if not cycle p:
980   result.b:=pen_cap(predir infinity of result.l,postdir 0 of result.r,
981     -postdir 0 of p,point 0 of p,local_nib_.l(last),local_nib_.r(0));
982   result.e:=pen_cap(predir infinity of result.r,postdir 0 of result.l,
983     predir last of p,point last of p,local_nib_.r(last), local_nib_.l(0));
984   result:=result.r && result.e && result.l && result.b && cycle;
985 fi
986 endif;
987
988 vardef pen_cap(expr a,b,c,p,niba,nibb)=
989   if path_eq(niba,nibb): pen_join(a,b,p,niba)
990   else: pen_join(a,c rotated 90,p,niba)-pen_join(c rotated 90,b,p,nibb)
991   fi
992 endif;
993
994
995

```

## Postscript Font Generation

```
996 % POSTSCRIPT FONT GENERATION
997
998 % Note that this has been stripped down a lot from the METATYPE1
999 % original code; most of the stuff for hinting, ligature tables,
1000 % METAFONT-style proof generation, and so on has been removed
1001 % because it's irrelevant to Tsukurimashou.
1002
1003 vardef pfi_file = jobname & ".pfi" enddef;
1004 vardef pic_file = "piclist" enddef;
1005 vardef dim_file = jobname & ".dim" enddef;
1006
1007 errorstopmode; warningcheck:=-1;
1008 ignore:=whatever; process:=0; utilize:=1; store:=2; % constants for introducing
1009 let semicolon_:=; ; % stores original meaning of a semicolon
1010 newinternal tracingdimens; % if ltracingdimens>0 then ldim_file is generated
1011
1012 def write_special = % additional info to be processed by AWK
1013   special "%GLYNFO: " &
1014 enddef;
1015 vardef mtone_glyph_pfx = "MT1: glyph " & str glyph_name & ": " enddef;
1016 def mtone_message = message mtone_glyph_pfx & enddef;
1017
1018 % Macro write_tex provides contact with the
1019 % outer world. The macro contains the information about EPSes that is
1020 % used for proofing and assembling the font; must be consistent with
1021 % the definitions contained in the files 'mpform.sty' and 'mp2pf.awk'.
1022 vardef write_tex(expr name, num) =
1023   write "\EPSNAMEandNUMBER{" & name & "}" & decimal(num) & "}"
1024   to pic_file & ".tex"
1025 enddef;
1026
1027 % The following macros are related to the operation of slanting.
1028 % In particular, they enable to keep a fixed width of a stem
1029 % after slanting.
1030 vardef slant_ang = % should be rather called "local_slant_angle"
1031   slang \ if known glyph_slanting.glyph_name: * glyph_slanting.glyph_name fi
1032 enddef;
1033 vardef slant_val = tand(slant_ang) enddef;
1034 vardef slant_preadjust(expr slope, slang) =
1035   % if sind(angle(slope))=0: 1 else:
1036   % | abs(sind(angle(slope))/sind(angle(cotd(angle(slope))+tand(slang),1)))|
1037   % |fi|
1038   % Correction of stem size taking into account its slope and a slant angle;
1039   % nice formula, isn't it? Much simpler than the previous one, yet equivalent:
1040   length(unitvector(slope) slanted tand(slang))
1041 enddef;
```

```

1042 vardef slant_stroke_val = slant_val enddef; % compatibility with plain_ex.mp
1043
1044 vardef stem_corr (expr slope) = slant_preadjust(slope, slant_ang) enddef;
1045
1046 def italicized = % fairly complex operation
1047   if slang<>0:
1048     if known glyph_slanting.glyph_name:
1049       if glyph_slanting.glyph_name=0: shifted (math_axis*tand(slang),0) fi
1050     fi
1051     shifted (italic_shift*tand(slang),0) % re-positioning
1052     slanted slant_val % and slanting
1053   fi
1054 enddef;
1055
1056 primarydef b || c =
1057   whatever*b + c*stem_corr(b)*unitvector(b rotated 90)
1058 enddef;
1059
1060 primarydef c /\ b =
1061   % A variant of the l|l procedure that iteratively counteracts slant
1062   % deformation; as with l|l, given: |c| – hypotenuse (vector) of
1063   % a right-angled triangle, |b| – the length of one of its legs;
1064   % result: the other leg of the triangle (vector),
1065   if slant_ang=0: (c leg b)
1066   else:
1067     begingroup save b_, b__, n_; b_:=b__:=b; n_:=10;
1068     forever:
1069       b_:=b*stem_corr(c leg b_);
1070       exitif (abs(b_-b__)<.01) or (n_<=0);
1071       b__:=b_; n_:=n_-1;
1072     endfor
1073     if (abs(b_-b__)>=.01):
1074       errhelp "The result is likely to be weird.";
1075       errmessage mtone_glyph_pfx & "iteration hasn't converged";
1076     fi
1077     c leg b_
1078   endgroup
1079 fi
1080 enddef;
1081
1082 % Obsolete?
1083 vardef rib(expr t,p,r) text u = % |u| is either empty or a vector
1084   save k_; pair k_; for i_:=u: k_:=u; endfor
1085   if unknown k_: k_=((udir t of p) rotated 90); fi
1086   (point t of p) + r * k_ * stem_corr(k_ rotated 90)
1087 enddef;
1088
1089 % The operation {\it compose_path\} is useful in \MP{} programs

```

```

1090 % automatically generated from PFB sources (pf2mt1 utility). Suffixes
1091 % $a$ and $b$ of control nodes stand for 'after' and 'before', respectively;
1092 % The operation {\it compose_path\}/} makes use of the operation
1093 % {\it compose_segment\}/} that serves for constructing non-cyclic
1094 % paths. Undefined nodes are ignored.
1095 vardef compose_segment@#(expr m,n) = % |m|<=n|, not checked
1096   if unknown inside_compose_path_: save idx_, n_; n_:=n-1; fi
1097   save n__; n__=n_+1;
1098   for i_:=m upto n: if known @#[i_]: idx_[incr(n_)]=i_; fi endfor
1099   for i_:=n__ upto n_+1:
1100     @#[idx_[i_]] .. controls
1101       @#[idx_[i_]] if known @#[idx_[i_]]a: a fi
1102       and @#[idx_[i_+1]] if known @#[idx_[i_+1]]b: b fi
1103     ..
1104   endfor
1105   @#[idx_[n_]]
1106 enddef;
1107 vardef compose_path@#(expr n) =
1108   save inside_compose_path_, idx_, n_; n_:=n-1; inside_compose_path_:=1;
1109   compose_segment@#(0,n)
1110   if @#[idx_[0]]=@#[idx_[n_]]: & else: - fi \ cycle
1111 enddef;
1112
1113 % Basic macros for building character glyphs:
1114 vardef round_node_values(expr p) =
1115   save d_; % candidates for Flex - no checking for "straightlinessness"
1116   for t_=0 upto length(p)-1:
1117     if round(point t_ of p)=round(point t_+1 of p):
1118       hide(mtone_message "degenerated bezier " & " ; length=" &
1119         decimal(length(p)) & " " & " ; time=" & decimal(t_) & " ";
1120       show p)
1121     else:
1122       round(point t_ of p)..
1123       if if known d_[t_] or known d_[t_+1]: false else:
1124         is_line(subpath (t_,t_+1) of p) fi:
1125         controls round(point t_ of p) and round(point t_+1 of p)
1126       else:
1127         controls round(postcontrol t_ of p) and round(precontrol t_+1 of p)
1128       fi
1129     ..
1130   fi
1131 endfor
1132 round(point length(p) of p) \ cycle p: & cycle fi
1133 enddef;
1134
1135 primarydef a start b =
1136   if cycle a:
1137     if b=default: default_start_(a)

```

```

1138 else: ((subpath (b,length(a)+b) of a) & cycle) fi
1139 else: a fi
1140 enddef;
1141
1142 newinternal default; default:=infinity;
1143 vardef default_start_(expr p) =
1144   save i_,j_,pi_,pj_; pair pi_,pj_;
1145   j_:=0; pj_:=point j_ of p;
1146   for i_=1 upto length(p):
1147     pi_:=point i_ of p;
1148     if (xpart(pi_)>xpart(pj_)) or
1149       (xpart(pi_)=xpart(pj_) and (ypart(pi_)<ypart(pj_)):
1150       j_:=i_; pj_:=point j_ of p;
1151     fi
1152   endfor
1153   (subpath (j_, length(p)+j_) of p) & cycle
1154 enddef;
1155
1156 def Fill text glist =
1157   begingroup
1158   save h_; path h_;
1159   for g_:=glist:
1160     h_:=g_ start.default; % JMN's suggestion
1161     if turningnumber h_<>1:
1162       errhelp "The result is likely to be weird.";
1163       errmessage mtone_glyph_pfx & "strange turning number in Fill, " &
1164         decimal(turningnumber h_);
1165     fi
1166     if glyph_usage div store = 1: % storing
1167       glyph_stored.glyph_name[incr glyph_stored.glyph_name.num]=h_;
1168     fi
1169     glyph_list[incr glyph_list.num]:=round_node_values(h_ italicized);
1170     update_glyph_bb(glyph_list[glyph_list.num]);
1171   endfor;
1172 endgroup
1173 enddef;
1174
1175 def unFill text glist =
1176   begingroup
1177   save h_; path h_;
1178   for g_:=glist:
1179     h_:=g_ start.default; % JMN's suggestion
1180     if turningnumber h_<>-1:
1181       errhelp "The result is likely to be weird.";
1182       errmessage mtone_glyph_pfx & "strange turning number in unFill, " &
1183         decimal(turningnumber h_);
1184     fi
1185     if glyph_usage div store = 1: % storing

```

```

1186     glyph_stored.glyph_name[incr glyph_stored.glyph_name.num]=h_;
1187     fi
1188     glyph_list[incr glyph_list.num]:=round_node_values(h_ italicized);
1189 endfor;
1190 endgroup
1191 enddef;
1192
1193 def fix_hsbw (expr xr,ml,mr) =
1194     glyph_shift:=round(ml); % shift = left margin
1195     glyph_width:=round(xr+ml+mr); % declared width plus margins
1196     if glyph_usage div store = 1: % storing
1197         glyph_shift.glyph_name:=glyph_shift; glyph_width.glyph_name:=glyph_width;
1198     fi
1199 enddef;
1200
1201 def fix_exact_hsbw(expr xr,ml,mr) =
1202     glyph_shift:=round(ml); % shift = left margin
1203     glyph_width:=xr+ml+mr; % declared width plus margins
1204     if glyph_usage div store = 1: % storing
1205         glyph_shift.glyph_name:=glyph_shift; glyph_width.glyph_name:=glyph_width;
1206     fi
1207 enddef;
1208
1209 % Macros below set PostScript and \TeX{} units; a trick with '\#'
1210 % in {\it tfm\_units\}/} proves useful in achieving compatibility
1211 % with the Knuthian fonts (e.g., it is employed in {\it logo\}/} font).
1212 % Old versions of {\it tfm\_units\}/} and {\it ps\_units\}/} are less
1213 % accurate, but are kept because of backward compatibility reasons.
1214 vardef tfm_units(text x) =
1215     save #; if known (x#): x# else: x/(1000/designsize) fi
1216 enddef;
1217 vardef old_tfm_units(text x) =
1218     save #; if known (x#): x# else: x/1000*designsize fi
1219 enddef;
1220
1221 vardef ps_units(expr x) = x*(1000/designsize) enddef;
1222 vardef old_ps_units(expr x) = x/designsize*1000 enddef;
1223
1224 def define_ps_units(text t) =
1225     forsuffices $:=t: $:=ps_units($. #); endfor
1226 enddef;
1227 def define_whole_ps_units(text t) =
1228     forsuffices $:=t: $:=round(ps_units($. #)); endfor
1229 enddef;
1230 def define_even_ps_units(text t) =
1231     forsuffices $:=t: $:=2round(1/2ps_units($. #)); endfor
1232 enddef;
1233

```

```

1234 % In general, all objects are supposed to be drawn by the
1235 % {\bf endglyph} macro, i.e., all drawing operations are deferred.
1236 % The same concerns labelling, which necessitates redefinition
1237 % of labelling macros.
1238
1239 def label_(suffix pos)(expr s,z, dot_or_not) =
1240 % should be more complex if overlapping labels are to be avoided
1241 enddef;
1242 string label_defaultfont; label_defaultfont:="cmr10";
1243 newinternal label_defaultscales; label_defaultscales:=magstep 5;
1244
1245 % If the {\it project\}/ variable is assigned value greater than 0,
1246 % proofing mode is assumed; the following macros display then
1247 % the details of the construction of glyphs for proofing purposes.
1248 % The larger value of the variable {\it project}, the more details
1249 % are visualised.
1250 def local_drawoptions (text t) = % to be used within a group, see below
1251 % \begingroup \def\#1{\it#1}% local: no underscore hacks
1252 save __op__; drawoptions(t);
1253 % \endgroup
1254 enddef;
1255
1256 def update_glyph_bb(expr p) =
1257 if unknown glyph_llx:
1258   glyph_llx:=xpart(llcorner(p)); glyph_lly:=ypart(llcorner(p));
1259   glyph_urx:=xpart(urcorner(p)); glyph_ury:=ypart(urcorner(p));
1260 else:
1261   if xpart(llcorner(p))<glyph_llx: glyph_llx:=xpart(llcorner(p)); fi
1262   if ypart(llcorner(p))<glyph_lly: glyph_lly:=ypart(llcorner(p)); fi
1263   if xpart(urcorner(p))>glyph_urx: glyph_urx:=xpart(urcorner(p)); fi
1264   if ypart(urcorner(p))>glyph_ury: glyph_ury:=ypart(urcorner(p)); fi
1265 fi
1266 enddef;
1267 string stencil_dir;
1268 def ship_glyphs =
1269   begingroup
1270     local_drawoptions();
1271     for g_:=1 upto glyph_list.num:
1272       if turningnumber glyph_list[g_]>0: fill else: unfill fi
1273       glyph_list[g_] shifted (glyph_shift,0);
1274     endfor
1275   endgroup
1276 enddef;
1277 newinternal show_stroke_size; show_stroke_size:=1.5;
1278 color show_stroke_color; show_stroke_color:=red;
1279
1280 color label_dot_color, label_text_color;
1281 label_dot_color:=.8white; label_text_color:=black;

```

```

1282 newinternal label_dot_size; label_dot_size:=3bp;
1283
1284 % Begin and end of the definitions of a character glyph:
1285 def begin_skip =
1286   let endglyph = fi;
1287   let ; = end_skip semicolon__
1288   if false:
1289   endif;
1290 def end_skip =
1291   let ; = semicolon__ semicolon__
1292   let endglyph = endglyph__;
1293   endif;
1294
1295 def uni_name(text name) = % name is either a suffix or a string expression
1296   if is_suffix(name):
1297     name
1298   else:
1299     scantokens (begingroup
1300       save rval;
1301       string rval;
1302       rval:="" for i=1 upto length (name):
1303         & "_" & (substring (i-1,i) of (name))
1304       endfor;
1305       rval
1306     endgroup)
1307   fi
1308   endif;
1309
1310 def glyph_name_ext = endif;
1311 def beginglyph(text name) =
1312   %
1313   def original_glyph_name = name endif;
1314   def glyph_name = uni_name(name) glyph_name_ext endif; % to use in lendglyph
1315   numeric glyph_usage; glyph_usage:=glyph_usage.glyph_name;
1316   if unknown glyph_usage: expandafter begin_skip fi
1317   string ps_name; ps_name:=original_glyph_name;
1318   if unknown ps_name:
1319     errhelp "Use macro 'introduce' or 'assign_name' prior to 'beginglyph.'";
1320     errmessage "MT1: PS name not assigned to " & str glyph_name;
1321   fi
1322   if name_used(original_glyph_name):
1323     errhelp "Proceed if you wish, I'll use the second glyph description.";
1324     errmessage "MT1: double output: name " & (str glyph_name);
1325   fi
1326   if glyph_usage mod store = 1: % utilizing
1327     mark_name_used(original_glyph_name);
1328   fi
1329   numeric glyph_code, glyph_num; glyph_code:=name_to_code(original_glyph_name);

```

```

1330 if glyph_code<0: glyph_num:=500-decr(min_glyph_code); else:
1331     glyph_num:=100+glyph_code;
1332     if glyph_code>max_glyph_code: max_glyph_code:=glyph_code; fi
1333 fi
1334 %
1335 beginfig(glyph_num)
1336 if glyph_usage mod store = 1: % utilizing
1337     write_special "NAME " & ps_name & " " & decimal(glyph_code);
1338     % mpform.sty and mp2pf.awk interface
1339 % lwrite_tex(glyph_name, glyph_num);l
1340     write_tex(ps_name, glyph_num);
1341 fi;
1342 glyph_list.num:=label_list.num:=0;
1343 path glyph_list[\\];
1344 picture label_list[\\]; pair label_list.dot[\\];
1345 numeric glyph_llx, glyph_lly, glyph_urx, glyph_ury;
1346 numeric bitmap_scale; pair bitmap_offset;
1347 numeric glyph_shift, glyph_width, glyph_axis;
1348 save glyph;
1349 hstem_list.num:=vstem_list.num:=hstem_list.cov:=vstem_list.cov:=0;
1350 pair hstem_list[\\], vstem_list[\\];
1351 path hstem_list_segms[\\], vstem_list_segms[\\];
1352 numeric old_hinting_scheme, new_hinting_scheme;
1353 if glyph_usage div store = 1: % storing
1354     if not path glyph_stored.glyph_name[0]: % glyph_name may contain digits
1355         scantokens("path " & genericize(str glyph_stored.glyph_name) & "[ ]");
1356         scantokens("pair " & genericize(str hstem_stored.glyph_name) & "[ ]");
1357         scantokens("path " & genericize(str hstem_stored_segms.glyph_name) & "[ ]");
1358         scantokens("pair " & genericize(str vstem_stored.glyph_name) & "[ ]");
1359         scantokens("path " & genericize(str vstem_stored_segms.glyph_name) & "[ ]");
1360     fi
1361     glyph_stored.glyph_name.num:=0;
1362     hstem_stored.glyph_name.num:=0; vstem_stored.glyph_name.num:=0;
1363 fi
1364 scantokens extra_beginglyph;
1365 enddef;
1366
1367 picture endglyph_picture;
1368 def endglyph =
1369     scantokens extra_endglyph;
1370     % usually, lcurrentpicture=nullpicturel, but if not (i.e., some
1371     % extra objects have been drawn), the picture must be shifted:
1372     endglyph_picture:=currentpicture shifted (glyph_shift,0);
1373     currentpicture:=nullpicture;
1374     if known glyph_axis: % actually, used only with stored chars
1375         glyph_axis.glyph_name:=glyph_axis;
1376     fi
1377 % fix char dimensions and write them to TFM and/or ldim_filel

```

```

1378 % independently of lglyph_usaget (ldim_filel)
1379 % fix_tfm_data(glyph_urx+glyph_shift, glyph_ury);
1380 if glyph_usage mod store = 1: % utilizing
1381     write_special "HSBW * " & decimal(glyph_width);
1382     write_special "BEGINCHAR";
1383     ship_glyphs;
1384 endfig;
1385 else:
1386 endgroup; % ends figure without shipping it out
1387 fi
1388 endif;
1389 let endglyph_=endglyph;
1390 string extra_beginglyph, extra_endglyph; extra_beginglyph=extra_endglyph="";
1391
1392 % Additional macros
1393 vardef fix_name_list text t =
1394 string name_list[]; numeric name_list.num; name_list.num:=0;
1395 save , ; let , = fix_name_list_; fix_name_list_ t
1396 endif;
1397 def fix_name_list_ suffix name =
1398 ; % important semicolon!
1399 if str name<>"": fix_name_list_s_ name else: fix_name_list_e_ "" & fi
1400 endif;
1401 def fix_name_list_s_ suffix s_name = fix_name_list_e_ (str s_name) endif;
1402 def fix_name_list_e_ expr e_name = % name is expected to be of the string type
1403 name_list[incr name_list.num]=e_name
1404 endif;
1405
1406 def introduce suffix name =
1407 if str name="": introduce_
1408 elseif (substring (0,1) of str name)<>"_": introduce__ name
1409 else: introduce___ name fi
1410 endif;
1411 def introduce_ expr name = % name is expected to be a string expression
1412 introduce___ uni_name(name)
1413 endif;
1414 vardef introduce___@#(expr usage, slanting)(text stencil) =
1415 if (unknown process_selected or known process_selected@#)
1416 and known usage and unknown ignore_selected@#:
1417 glyph_usage@#:=usage; % lignore=whateverl, lprocess=0l, lutilize=1l, lstore=2l
1418 if unknown glyph_ps_name@#: % set default:
1419 assign_name @# (substring (1,infinity) of (str @#));
1420 fi
1421 glyph_slanting@#:=slanting; % ignore lslant_angl if l0l; use lslant_angl otherwise
1422 % lstencil can be either string (recommended) or suffix (with default
1423 % extension l".eps"l - obsolete), hence some trickery below
1424 save r_; string r_;
1425 for i_:=stencil: if string i_: r_:=i_; fi endfor

```

```

1426   if unknown r_:
1427     forsuffices i_:=stencil: r_:= str i_; endfor
1428     if r_<>"": r_:=r_ & ".eps"; fi
1429   fi
1430   if r_<>"":
1431     if not string glyph_stencil@#:
1432       scantokens("string " & genericize(str glyph_stencil@#));
1433     fi
1434     glyph_stencil@# = r_;
1435   fi
1436 fi
1437 endif;
1438
1439 vardef assign_name@#(expr ps_name) =
1440 if not string glyph_ps_name @#:
1441   scantokens("string " & genericize(str glyph_ps_name@#));
1442 fi
1443 glyph_ps_name@#:=ps_name;
1444 endif;
1445
1446 def standard_introduce(expr name) =
1447   introduce name (utilize+store)(1)();
1448 endif;
1449
1450 vardef name_to_code(text name) =
1451   save res_, name_; string name_;
1452   name_:=name; res_=-1;
1453   for i:=0 upto 255: % 1-to-1 coding presumed
1454     if known code_to_name_[i]: if code_to_name_[i]=name_: res_:=i; fi fi
1455     exitif res_>-1;
1456   endfor
1457   res_
1458 endif;
1459
1459 def encode(text name)(expr glyph_code)=
1460 if (glyph_code<0) or (glyph_code>255):
1461   errhelp "The code must be within the range 0..255.";
1462   errmessage "MT1: improper code " & decimal(glyph_code) &
1463     " ('encode' ignored)";
1464 elseif known code_to_name_[glyph_code]:
1465   errhelp "A given code can be assigned only to one name (obviously).";
1466   errmessage "MT1: repeated code for " & code_to_name_[glyph_code] &
1467     " (" & decimal(glyph_code) & "; 'encode' ignored)";
1468 else:
1469   code_to_name_[glyph_code]:=name;
1470 fi
1471 endif;
1472 string code_to_name_[\];
1473

```

```

1474 vardef name_used(text name) =
1475   save res_, name_; boolean res_; string name_;
1476   name_:=name; res_:=false;
1477   for i:=1 upto max_name_used: res_:=(name_used_[i]=name_); exitif res_; endfor
1478   res_
1479 enddef;
1480 def mark_name_used(text name)=
1481   name_used_[incr max_name_used]:=name;
1482 enddef;
1483 string name_used_["\"]; newinternal max_name_used;
1484
1485 vardef string_date =
1486   if day<10: "0" & fi decimal(day) & ":" &
1487   if month<10: "0" & fi decimal(month) & ":" &
1488   decimal(year)
1489 enddef;
1490
1491 def set_pfi (suffix kind) (expr val) =
1492   if known val:
1493     if (numeric val) or (string val) or (boolean val):
1494       if (numeric val) and (not numeric pf_info_set.kind):
1495         scantokens ("numeric " & genericize(str pf_info_set.kind));
1496       elseif (string val) and (not string pf_info_set.kind):
1497         scantokens ("string " & genericize(str pf_info_set.kind));
1498       elseif (boolean val) and (not boolean pf_info_set.kind):
1499         scantokens ("boolean " & genericize(str pf_info_set.kind));
1500       fi
1501       pf_info_set.kind:=val;
1502       write str kind & " : " &
1503       if string val: val
1504       elseif numeric val: decimal(val)
1505       elseif boolean val: if val: "true" else: "false" fi
1506       fi
1507       to pfi_file;
1508     else:
1509       errhelp "Proceed, I'll just ignore the setting.";
1510       errmessage "MT1: pf_info keys can only be of numeric, string " &
1511       "and boolean type";
1512     fi
1513   fi
1514 enddef;
1515
1516 def pf_info_version expr v = set_pfi(VERSION,v); enddef;
1517
1518 def pf_info_creationdate text t =
1519   begingroup
1520   save k_; k_:=0;
1521   for t_:=t: k_:=k_+1; set_pfi(CREATION_DATE, t_); exitif k_=1; endfor

```

```

1522 if k_=0: set_pfi(CREATION_DATE, string_date); fi
1523 endgroup
1524 enddef;
1525
1526 def pf_info_fontname text t =
1527   begingroup
1528     save k_; k_:=0;
1529     for t_:=t: k_:=k_+1;
1530       if k_=1: set_pfi(FONT_NAME, t_); fi
1531       if k_=2: set_pfi(FULL_NAME, t_); fi
1532       exitif k_=2;
1533     endfor
1534     if k_=1: set_pfi(FULL_NAME, pf_info_set.FONT_NAME); fi
1535   endgroup
1536 enddef;
1537
1538 def pf_info_author expr v = set_pfi(AUTHOR,v); enddef;
1539 % There is 'much ado about nothing,' i.e., about the sign of descender:
1540 % in a PFB file in an 'ADL' comment, descender is positive, while in an AFM
1541 % in a 'Descender' comment – negative; we will distinguish between
1542 % the two, the more so as 'ADL' comment is not mentioned in
1543 % in the Adobe documentation {\it Adobe Type 1 Font Format}.
1544
1545 def pf_info_ascender expr v = ascender:=v; set_pfi(ASCENDER,v); enddef;
1546 def pf_info_descender expr v = descender:=v; set_pfi(DSCENDER,v); enddef;
1547
1548 def pf_info_adl text t =
1549   begingroup
1550     save k_; k_:=0;
1551     for t_:=t: k_:=k_+1;
1552       if (k_=1) and known t_: adl_ascender:=t_; set_pfi(ADL_ASCENDER,t_); fi
1553       if (k_=2) and known t_: adl_descender:=t_; set_pfi(ADL_DESCENDER,t_); fi
1554       if (k_=3) and known t_: adl_lineskip:=t_; set_pfi(ADL_LINESKIP,t_); fi
1555       exitif k_=3;
1556     endfor
1557   endgroup
1558 enddef;
1559
1560 def pf_info_underline text t =
1561   begingroup
1562     save k_; k_:=0;
1563     for t_:=t: k_:=k_+1;
1564       if k_=1: set_pfi(UNDERLINE_POSITION,t_); fi
1565       if k_=2: set_pfi(UNDERLINE_THICKNESS,t_); fi
1566       exitif k_=2;
1567     endfor
1568   endgroup
1569 enddef;

```

```

1570
1571 def pf_info_pfm text t =
1572 % parameters: name, bold (0 or 1), italic (0 or 1), char set;
1573 % each of them can be either known or unknown or "*" (which means unknown);
1574 % the last parameter can be either numeric or string representation of
1575 % a valid Perl numeric value (e.g., "0xFF" means 255).
1576 begingroup
1577   save k_; k_:=0;
1578   for t_:=t: k_:=k_+1;
1579     if (k_==1) and known t_: set_pfi(PFM_NAME,t_); fi
1580     if (k_==2) and known t_: set_pfi(PFM_BOLD,t_); fi
1581     if (k_==3) and known t_: set_pfi(PFM_ITALIC,t_); fi
1582     if (k_==4) and known t_: set_pfi(PFM_CHARSET,t_); fi
1583     exitif k_==4;
1584   endfor
1585 endgroup
1586 enddef;
1587
1588 def pf_info_fixedpitch expr v = set_pfi(FIXED_PITCH,v); enddef;
1589 def pf_info_capheight expr v = uc_height:=v; set_pfi(CAPHEIGHT,v); enddef;
1590 def pf_info_weight expr v = set_pfi(WEIGHT,v); enddef;
1591 def pf_info_stdvstem expr v = set_pfi(STDVW,v); enddef;
1592 def pf_info_stdhstem expr v = set_pfi(STDHW,v); enddef;
1593 def pf_info_forcebold expr v = set_pfi(FORCE_BOLD,v); enddef;
1594
1595 % TeX-related font info (fontdimens and headerbytes):
1596 def pf_info_fontdimen text t = % exceptionally, TFM units expected
1597 begingroup
1598   save i_, k_, b_; boolean b_;
1599   k_:=0;
1600   if true for t_:=t: hide(k_:=k_+1) and known t_ endfor and (k_<=3):
1601     k_:=0;
1602     for t_:=t: k_:=k_+1;
1603       if k_==1:
1604         i_:=t_;
1605         % |b| means "we are ready to override (possibly) the previous value
1606         % of a font parameter unless we are inside |complete_tfm_infol and
1607         % then we want to set only a 'virgin' value."
1608         b_:=unknown completing_tfm_info or unknown pf_info_set.FONT_DIMEN[i_];
1609       fi
1610       if b_ and (k_==2): set_pfi(FONT_DIMEN[i_],t_); fontdimen i_: t_; fi
1611       if b_ and (k_==3): set_pfi(DIMEN_NAME[i_],t_); fi
1612     endfor
1613     if b_ and (k_==2): set_pfi(DIMEN_NAME[i_],"(unknown fontdimen name)"); fi
1614   else:
1615     errhelp "Proceed, I'll just ignore TFM fontdimen settings.";
1616     errmessage "MT1: invalid TFM fontdimen data";
1617   fi

```

```

1618 endgroup
1619 enddef;
1620 def pf_info_headerbyte text t =
1621   begingroup
1622   save i_, k_; k_:=0;
1623   if true for t_:=t: hide(k_:=k_+1) and known t_ endfor and (k_:=2):
1624     k_:=0;
1625     for t_:=t: k_:=k_+1;
1626       if k_:=1: i_:=t_; fi
1627       if k_:=2:
1628         set_pfi(HEADER_BYTE[i_],if numeric t_: decimal(t_) else: t_ fi);
1629         if i_:=9: % encoding scheme, e.g., "TEX TEXT"
1630           headerbyte 9: BCPL_string(t_,40); fi
1631         if i_:=49: % font family, e.g., "CMR"
1632           headerbyte 49: BCPL_string(t_,20); fi
1633         if i_:=72: % family member number, which should be between 0 and 255
1634           headerbyte 72: t_; fi
1635       fi
1636     endfor
1637   else:
1638     errhelp "Proceed, I'll just ignore TFM headerbyte settings.";
1639     errmessage "MT1: invalid TFM headerbyte data";
1640   fi
1641 endgroup
1642 enddef;
1643 def pf_info_designsize expr v = % ldesignsize is special
1644   designsize:=v; set_pfi(DSIGN_SIZE,decimal(v) & " (in points)");
1645 enddef;
1646 def pf_info_italicangle expr v =
1647   begingroup
1648   save tfm_units; vardef tfm_units(text x) = c enddef;
1649   slang:=v; set_pfi(ITALIC_ANGLE,v);
1650   pf_info_fontdimen 1, if known slant: slant else: tand(slang) fi, "(slant)";
1651 endgroup
1652 enddef;
1653 def pf_info_space text t = % three in one
1654   begingroup
1655   save k_; k_:=0;
1656   for t_:=t: k_:=k_+1;
1657     if (designsize<>0) and known t_:
1658       if k_:=1:
1659         space:=t_; pf_info_fontdimen 2, tfm_units(space), "(space)";
1660       elseif k_:=2:
1661         space_stretch:=t_; pf_info_fontdimen 3, tfm_units(space_stretch),
1662           "(space stretch)";
1663       elseif k_:=3:
1664         space_shrink:=t_; pf_info_fontdimen 4, tfm_units(space_shrink),
1665           "(space shrink)";

```

```

1666     fi
1667     fi
1668     exitif k_ = 3;
1669     endfor
1670 endgroup
1671 enddef;
1672 def pf_info_normal_space text t =
1673   begingroup
1674     save k_; k_ := 0;
1675     if true for t_ := t: hide(k_ := k_ + 1) endfor and (k_ <= 2):
1676       k_ := 0;
1677       for t_ := t: k_ := k_ + 1;
1678         if (k_ = 1) and known t_: space := t_; fi
1679         if (k_ = 2) and known t_: % |t_| is expected to be in TFM units
1680           pf_info_fontdimen 2, t_, "(space)";
1681         fi
1682       endfor
1683       if (k_ = 1) and (designsize <> 0) and known space:
1684         pf_info_fontdimen 2, tfm_units(space), "(space)";
1685       fi
1686     fi
1687 endgroup
1688 enddef;
1689 def pf_info_space_stretch text t =
1690   begingroup
1691     save k_; k_ := 0;
1692     if true for t_ := t: hide(k_ := k_ + 1) endfor and (k_ <= 2):
1693       k_ := 0;
1694       for t_ := t: k_ := k_ + 1;
1695         if (k_ = 1) and known t_: space_stretch := t_; fi
1696         if (k_ = 2) and known t_: % |t_| is expected to be in TFM units
1697           pf_info_fontdimen 3, t_, "(space stretch)";
1698         fi
1699       endfor
1700       if (k_ = 1) and (designsize <> 0) and known space_stretch:
1701         pf_info_fontdimen 3, tfm_units(space_stretch), "(space stretch)";
1702       fi
1703     fi
1704 endgroup
1705 enddef;
1706 def pf_info_space_shrink text t =
1707   begingroup
1708     save k_; k_ := 0;
1709     if true for t_ := t: hide(k_ := k_ + 1) endfor and (k_ <= 2):
1710       k_ := 0;
1711       for t_ := t: k_ := k_ + 1;
1712         if (k_ = 1) and known t_: space_shrink := t_; fi
1713         if (k_ = 2) and known t_: % |t_| is expected to be in TFM units

```

```

1714     pf_info_fontdimen 4, t_, "(space shrink)";
1715     fi
1716   endfor
1717   if (k_=1) and (designsize<>0) and known space_shrink:
1718     pf_info_fontdimen 4, tfm_units(space_shrink), "(space shrink)";
1719   fi
1720 fi
1721 endgroup
1722 enddef;
1723 def pf_info_xheight text t =
1724   begingroup
1725   save k_; k_:=0;
1726   if true for t_:=t: hide(k_:=k_+1) endfor and (k_<=2):
1727     k_:=0;
1728     for t_:=t: k_:=k_+1;
1729       if (k_=1) and known t_: lc_height:=t_; set_pfi(XHEIGHT, t_); fi
1730       if (k_=2) and known t_: % |t_| is expected to be in TFM units
1731         pf_info_fontdimen 5, t_, "(xheight)";
1732       fi
1733     endfor
1734     if (k_=1) and (designsize<>0) and known lc_height:
1735       pf_info_fontdimen 5, tfm_units(lc_height), "(xheight)";
1736     fi
1737   fi
1738 endgroup
1739 enddef;
1740 def pf_info_quad text t =
1741   begingroup
1742   save k_; k_:=0;
1743   if true for t_:=t: hide(k_:=k_+1) endfor and (k_<=2):
1744     k_:=0;
1745     for t_:=t: k_:=k_+1;
1746       if (k_=1) and known t_: quad:=t_; fi
1747       if (k_=2) and known t_: % |t_| is expected to be in TFM units
1748         pf_info_fontdimen 6, t_, "(quad)";
1749       fi
1750     endfor
1751     if (k_=1) and (designsize<>0) and known quad:
1752       pf_info_fontdimen 6, tfm_units(quad), "(quad)";
1753     fi
1754   fi
1755 endgroup
1756 enddef;
1757 def pf_info_extra_space text t =
1758   begingroup
1759   save k_; k_:=0;
1760   if true for t_:=t: hide(k_:=k_+1) endfor and (k_<=2):
1761     k_:=0;

```

```

1762   for t_:=t: k_:=k_+1;
1763     if (k_=1) and known t_: extra_space:=t_; fi
1764     if (k_=2) and known t_: % |t_| is expected to be in TFM units
1765       pf_info_fontdimen 7, t_, "(extra space)";
1766     fi
1767   endfor
1768   if (k_=1) and (designsize<>0) and known extra_space:
1769     pf_info_fontdimen 7, tfm_units(extra_space), "(extra space)";
1770   fi
1771 fi
1772 endgroup
1773 endif;
1774 def pf_info_encoding text t =
1775   begingroup
1776     save k_; k_:=0;
1777     for t_:=t: k_:=k_+1;
1778       if (k_=1) and known t_: if t_<>"": set_pfi(ENCODING_SCHEME, t_); fi fi
1779       if (k_=2) and known t_: if t_<>"": pf_info_headerbyte 9, t_; fi fi
1780       if (k_=3) and known t_: if t_<>"": set_pfi(ENCODING_NAME, t_); fi fi
1781       exitif k_=3;
1782     endfor
1783     if (k_=1) and known pf_info_set.ENCODING_SCHEME % upward compatibility
1784       and unknown pf_info_set.HEADER_BYTE9:
1785       pf_info_headerbyte 9, pf_info_set.ENCODING_SCHEME;
1786     fi
1787   endgroup
1788 endif;
1789 def pf_info_familyname text t =
1790   begingroup
1791     save k_; k_:=0;
1792     for t_:=t: k_:=k_+1;
1793       if k_=1: set_pfi(FAMILY_NAME, t_); fi
1794       if k_=2: pf_info_headerbyte 49, t_; fi
1795       exitif k_=2;
1796     endfor
1797     if k_=1: pf_info_headerbyte 49, pf_info_set.FAMILY_NAME; fi
1798   endgroup
1799 endif;
1800
1801 % bluezz forever...
1802 newinternal blue_fuzz, blue_scale, blue_shift;
1803 blue_fuzz:=0; % Adobe Type 1 Font Format, p. 41
1804 blue_scale:=0.0454545;
1805 blue_shift:=7;
1806
1807 % it is advisable to avoid typso whenever possible:
1808 def show_compose expr x = show_compose_ :=x; enddef;
1809 def show_fills expr x = show_fills_ :=x; enddef;

```

```

1810 def show_strokes expr x = show_strokes_ :=x; enddef;
1811 def show_paths expr x = show_paths_ :=x; enddef;
1812 def show_labels expr x = show_labels_ :=x; enddef;
1813 def show_boxes expr x = show_boxes_ :=x; enddef;
1814 def show_stems expr x = show_stems_ :=x; enddef;
1815 def show_stencils expr x = show_stencils_ :=x; enddef;
1816
1817 string extra_beginfont, extra_endfont; extra_beginfont=extra_endfont="";
1818
1819 def beginfont =
1820   min_glyph_code=max_glyph_code=0;
1821   complete_param_setting;
1822   scantokens extra_beginfont;
1823 enddef;
1824
1825 def complete_param_setting =
1826   if designsize=0: designsize:=10; fi
1827   if unknown space: space:=333; fi
1828   if unknown space_stretch: space_stretch:=round(1/2space); fi
1829   if unknown space_shrink: space_shrink:=round(1/3space); fi
1830   if unknown extra_space: extra_space:=round(1/3space); fi
1831   if unknown quad: quad:=1000; fi
1832   if unknown slang:
1833     if known slant: % compatibility with the Old Tradition...
1834       slang:=angle(1, slant);
1835     else: slang:=0; fi
1836   fi
1837   if unknown uc_height: uc_height:=750; fi
1838   if unknown lc_height: lc_height:=400; fi
1839   if unknown italic_shift: italic_shift:=-40; fi % used to be l-100l
1840   if unknown depth: depth:=-250; fi
1841   if unknown ascender: ascender:=uc_height; fi
1842   if unknown descender: descender:=depth; fi
1843   if unknown adl_ascender: adl_ascender:=uc_height; fi
1844   if unknown adl_descender: adl_descender:=-depth; fi
1845   if unknown adl_lineskip: adl_lineskip:=0; fi
1846   if unknown top_line: top_line:=adl_ascender+1/2adl_lineskip; fi
1847   if unknown bot_line: bot_line:=-adl_descender+1/2adl_lineskip; fi
1848   if unknown math_axis: math_axis:=250; fi
1849   if unknown math_rule: math_rule:=40; fi
1850   begingroup
1851     save rth_, pt_, subs_, desc_depth_, fig_height_, asc_height_;
1852     rth_:=math_rule; pt_:=100;
1853     % math symbol font parameters (defaults excerpted from cmsy10)
1854     subs_:=7/10;
1855     desc_depth_:=70/36pt_; fig_height_:=232/36pt_; asc_height_:=250/36pt_;
1856     if unknown num_one:
1857       num_one:=math_axis+3.51rth_+54/36pt_+subs_*desc_depth_; fi

```

```

1858 if unknown num_two: num_two:=math_axis+1.5l_rth_*30/36pt_; fi
1859 if unknown num_three: num_three:=math_axis+1.5l_rth_*48/36pt_; fi
1860 if unknown denom_one:
1861     denom_one:=3.5l_rth_*subs_*fig_height_*124/36pt_-math_axis; fi
1862 if unknown denom_two:
1863     denom_two:=1.5l_rth_*subs_*fig_height_*30/36pt_-math_axis; fi
1864 if unknown sup_one: sup_one:=8.99pt_-subs_*asc_height_; fi
1865 if unknown sup_two: sup_two:=8.49pt_-subs_*asc_height_; fi
1866 if unknown sup_three: sup_three:=104/36pt_; fi
1867 if unknown sub_one: sub_one:=54/36pt_; fi
1868 if unknown sub_two: sub_two:=-8.49pt_*2subs_*asc_height_*3.l_rth_; fi
1869 if unknown sup_drop: sup_drop:=subs_*asc_height_-36/36pt_; fi
1870 if unknown sub_drop: sub_drop:=18/36pt_; fi
1871 if unknown delim_one: delim_one:=23.9pt_; fi
1872 if unknown delim_two: delim_two:=10.1pt_; fi
1873 % math extension font parameters (defaults excerpted from cmex10)
1874 if unknown big_op_spacing_one: big_op_spacing_one:=40/36pt_; fi
1875 if unknown big_op_spacing_two: big_op_spacing_two:=60/36pt_; fi
1876 if unknown big_op_spacing_three: big_op_spacing_three:=72/36pt_; fi
1877 if unknown big_op_spacing_four: big_op_spacing_four:=216/36pt_; fi
1878 if unknown big_op_spacing_five: big_op_spacing_five:=36/36pt_; fi
1879 endgroup;
1880 enddef;
1881
1882 def endfont =
1883     scantokens extra_endfont;
1884     complete_pf_info;
1885     complete_tfm_info;
1886     scantokens "end";
1887 enddef;
1888
1889 def complete_pf_info =
1890     if unknown pf_info_set.DESIGN_SIZE: pf_info_designsize designsize; fi
1891     if unknown pf_info_set.VERSION: pf_info_version "0.000"; fi
1892     if unknown pf_info_set.AUTHOR: pf_info_author "Unknown"; fi
1893     if unknown pf_info_set.CREATION_DATE: pf_info_creationdate; fi
1894     if unknown pf_info_set.FAMILY_NAME: pf_info_familyname "Untitled"; fi
1895     if unknown pf_info_set.FONT_NAME: pf_info_fontname "Untitled"; fi
1896     if unknown pf_info_set.ASCENDER: pf_info_ascender ascender; fi
1897     if unknown pf_info_set.DESCENDER: pf_info_descender descender; fi
1898     if unknown pf_info_set.ADL_ASCENDER:
1899         pf_info_adl adl_ascender, whatever, whatever;
1900     fi
1901     if unknown pf_info_set.ADL_DESCENDER:
1902         pf_info_adl whatever, adl_descender, whatever;
1903     fi
1904     if unknown pf_info_set.ADL_LINESKIP:
1905         pf_info_adl whatever, whatever, adl_lineskip;

```

```

1906 fi
1907 if unknown pf_info_set.UNDERLINE_POSITION: pf_info_underline -200, whatever;
1908     fi
1908 if unknown pf_info_set.UNDERLINE_THICKNESS: pf_info_underline whatever, math_rule;
1909     fi
1909 if unknown pf_info_set.ITALIC_ANGLE: pf_info_italicangle slang; fi
1910 if unknown pf_info_set.FIXED_PITCH: pf_info_fixedpitch false; fi
1911 if unknown pf_info_set.CAPHEIGHT: pf_info_capheight uc_height; fi
1912 if unknown pf_info_set.XHEIGHT: pf_info_xheight lc_height; fi
1913 if unknown pf_info_set.WEIGHT: pf_info_weight "Normal"; fi
1914 if unknown pf_info_set.STDVW: fi % just ignore
1915 if unknown pf_info_set.STDHW: fi % just ignore
1916 if unknown pf_info_set.FORCE_BOLD: pf_info_forcebold false; fi
1917 if unknown pf_info_set.ENCODING_SCHEME:
1918     pf_info_encoding "FontSpecific", whatever;
1919 fi
1920 if unknown pf_info_set.HEADER_BYTE9:
1921     pf_info_encoding whatever, "UNSPECIFIED";
1922 fi
1923 if unknown pf_info_set.BLUE_VALUES: set_pfi(BLUE_VALUES, ""); fi
1924 if unknown pf_info_set.OTHER_BLUES: fi % just ignore
1925 if unknown pf_info_set.BLUE_FUZZ: set_pfi(BLUE_FUZZ, blue_fuzz); fi
1926 if unknown pf_info_set.BLUE_SCALE: set_pfi(BLUE_SCALE, blue_scale); fi
1927 if unknown pf_info_set.BLUE_SHIFT: set_pfi(BLUE_SHIFT, blue_shift); fi
1928 % for those who like smart (implicit) systems:
1929 if unknown no_implicit_spaces:
1930     if not name_used("space"):
1931         if unknown glyph_usage._space: introduce _space (utilize)(0)(); fi
1932         if (name_to_code("space")<0) and (unknown code_to_name_32):
1933             encode("space") (32);
1934         fi
1935         beginglyph("_space") fix_hsbw(space,0,0); endglyph;
1936     fi
1937     if not name_used("nbspace"):
1938         if unknown glyph_usage._nbspace: introduce _nbspace (utilize)(0)(); fi
1939         %
1940         beginglyph("_nbspace") fix_hsbw(space,0,0); endglyph; % normal space width
1941     fi
1942 fi
1943 enddef;
1944
1945 def complete_tfm_info =
1946     % complete fontdimen info:
1947     % ldesignsizel is expected to be known
1948     % lslantl dimen has already been set; lheightl dimen – not necessarily,
1949     % but lpf_info_set.XHEIGHTl is known:
1950     completing_tfm_info:=1;
1951     pf_info_xheight whatever,

```

```

1952   if known lc_height#: lc_height# else: tfm_units(pf_info_set.XHEIGHT) fi;
1953 pf_info_normal_space space if known space#: , space# fi;
1954 pf_info_space_stretch space_stretch
1955   if known space_stretch#: , space_stretch# fi;
1956 pf_info_space_shrink space_shrink if known space_shrink#: , space_shrink# fi;
1957 pf_info_quad quad if known quad#: , quad# fi;
1958 pf_info_extra_space extra_space if known extra_space#: , extra_space# fi;
1959 font_math_rule math_rule;
1960 font_math_axis math_axis;
1961 % complete header info:
1962 pf_info_headerbyte 72, max(0, 254 - round 2designsize);
1963 completing_tfm_info:=whatever;
1964 endif;
1965
1966 def BCPL_string(expr s,n)= % string lsl becomes an lnl-byte BCPL string
1967   for l:=if length(s)>=n: n-1 else: length(s) fi: l
1968     for k:=1 upto l: , substring (k-1,k) of s endfor
1969     for k:=l+2 upto n: , 0 endfor endfor
1970 endif;
1971
1972 % The Old Tradition...
1973 def font_size expr x = designsize:=x endif;
1974 def font_slant expr x = fontdimen 1: x endif;
1975 def font_normal_space expr x = fontdimen 2: x endif;
1976 def font_normal_stretch expr x = fontdimen 3: x endif;
1977 def font_normal_shrink expr x = fontdimen 4: x endif;
1978 def font_x_height expr x = fontdimen 5: x endif;
1979 def font_quad expr x = fontdimen 6: x endif;
1980 def font_extra_space expr x = fontdimen 7: x endif;
1981
1982 % A New Tradition...
1983 def def_font_param (suffix param_name)(expr param_num, param_desc) =
1984   def param_name text x =
1985     begingroup save #; % cf. the definition of ltfm_unitsl
1986     if (known x#) or ((designsize<>0) and known x):
1987       pf_info_fontdimen param_num, tfm_units(x), "(" & param_desc & ")";
1988     fi
1989   endgroup
1990   endif;
1991 endif;
1992
1993 def_font_param (font_math_rule, 8, "math rule");
1994 def_font_param (font_math_axis, 22, "math axis");
1995 % symbol fonts
1996 def_font_param (font_num_one, 8, "num1");
1997 def_font_param (font_num_two, 9, "num2");
1998 def_font_param (font_num_three, 10, "num3");
1999 def_font_param (font_denom_one, 11, "denom1");

```

```

2000 def_font_param (font_denom_two, 12, "denom2");
2001 def_font_param (font_sup_one, 13, "sup1");
2002 def_font_param (font_sup_two, 14, "sup2");
2003 def_font_param (font_sup_three, 15, "sup3");
2004 def_font_param (font_sub_one, 16, "sub1");
2005 def_font_param (font_sub_two, 17, "sub2");
2006 def_font_param (font_sup_drop, 18, "sup_drop");
2007 def_font_param (font_sub_drop, 19, "sub_drop");
2008 def_font_param (font_delim_one, 20, "delim1");
2009 def_font_param (font_delim_two, 21, "delim2");
2010 % extension fonts
2011 def_font_param (font_big_op_spacing_one, 9, "big_op_spacing1");
2012 def_font_param (font_big_op_spacing_two, 10, "big_op_spacing2");
2013 def_font_param (font_big_op_spacing_three, 11, "big_op_spacing3");
2014 def_font_param (font_big_op_spacing_four, 12, "big_op_spacing4");
2015 def_font_param (font_big_op_spacing_five, 13, "big_op_spacing5");
2016
2017 endinput

```

## obstack.mp

```
1 %
2 % Object stack for Tsukurimashou
3 % Copyright (C) 2011, 2012 Matthew Skala
4 %
5-29 [Standard copyright notice]
30
31 inclusion__lock(obstack);
32
33 _____
34
```

### Object Stack Data

```
35 % OBJECT STACK DATA
36
37 % object types:
38 % "anchor" - uses transform, numeric
39 % "hook" - uses string, numeric
40 % "lcblob" - uses path p
41 % "null" - uses nothing
42 % "pbox" - uses transform
43 % "stroke" - uses path p, path q, numeric array, bool array
44
45 vardef init_obstack =
46   numeric obstacktype[];
47   numeric obstackn[];
48   numeric obstackna[][];
49   numeric obstacknaa[][][];
50   boolean obstackb[];
51   boolean obstackba[][];
52   path obstackp[];
53   path obstackq[];
54   transform obstackt[];
55   string obstacks[];
56   numeric sp;
57   sp:=1;
58 enddef;
59
60 sp:=0;
61
62 % numeric values, needed for syntax reasons
63
64 def boalternate = 1924 enddef;
65 def bokeepshape = 1838 enddef;
66 def boserif = 1746 enddef;
67 def bosize = 1393 enddef;
68 def botip = 1322 enddef;
```

**OBST**

```

69 def botoexpand = 1972 enddef;
70
71 def hsmain_render = 1304 enddef;
72
73 def otanchor = 1882 enddef;
74 def othook = 1753 enddef;
75 def otlcblob = 1722 enddef;
76 def otnull = 1699 enddef;
77 def otpbox = 1007 enddef;
78 def otstroke = 1069 enddef;
79
80
81

```

---

## Object Stack Methods

```

82 % OBJECT STACK METHODS
83
84 vardef expand_pbox =
85   begingroup
86     save mysp,i;
87     numeric mysp;
88     for i=sp-1 downto 1:
89       if (obstacktype[i]=otpbox) and (known obstackba.botoexpand[i]):
90         if obstackba.botoexpand[i]:
91           mysp:=i;
92           fi;
93         fi;
94       exitif known mysp;
95     endfor;
96     if known mysp:
97       obstackba.botoexpand[mysp]:=false;
98 % message "expanding " & decimal mysp;
99     save x,y,myxf;
100     numeric x[],y[];
101     transform myxf;
102     z1=(70,830);
103     z2=(930,-30);
104     for i=mysp+1 upto sp-1:
105       if obstacktype[i]=otpbox:
106         x3:=xpart ((0,0.5) transformed obstackt[i]);
107         if x3<x1: x1:=x3; fi;
108         y3:=ypart ((0.5,1) transformed obstackt[i]);
109         if y3>y1: y1:=y3; fi;
110         x4:=xpart ((1,0.5) transformed obstackt[i]);
111         if x4>x2: x2:=x4; fi;
112         y4:=ypart ((0.5,0) transformed obstackt[i]);
113         if y4<y2: y2:=y4; fi;

```

```

114         fi;
115     endfor;
116     z0=(x1,y2);
117     (0,0) transformed myxf=z0+(-20,-20);
118     (0,1) transformed myxf=z1+(-20,20);
119     (1,0) transformed myxf=z2+(20,-20);
120     obstackt[myp]:=myxf;
121 else:
122     errmessage "Can't find PBOX to expand";
123 fi;
124 endgroup;
125 perl_structure:=perl_structure&"]";
126 enddef;
127
128 vardef find_stroke(expr idx) =
129     (find_whatever(otstroke,idx))
130 enddef;
131
132 vardef find_whatever(expr w,idx) =
133     begingroup
134         save i,j;
135         numeric i,j;
136         i:=sp-1;
137         j:=-idx;
138         forever:
139             exitif i<=0;
140             if obstacktype[i]=w:
141                 j:=j-1;
142                 fi;
143             exitif j<0;
144             i:=i-1;
145         endfor;
146     i
147 endgroup
148 enddef;
149
150 vardef get_bosize(expr idx) =
151     obstackna.bosize[find_whatever(otstroke,idx)]
152 enddef;
153
154 vardef get_anchor_with_default(expr atype,default_anchor) =
155     begingroup
156         save i,j;
157         numeric i,j;
158         i:=0;
159         forever:
160             j:=find_whatever(otanchor,i);
161             exitif j<=0;

```

```

162     exitif obstackn[j]=atype;
163     i:=i-1;
164     endfor;
165     if j<=0: default_anchor else: obstackt[j] fi
166 endgroup
167 enddef;
168
169 vardef get_anchor(expr atype) =
170   get_anchor_with_default(atype,identity)
171 enddef;
172
173 vardef get_lcblob(expr idx) =
174   obstackp[find_whatever(otlcblob,idx)]
175 enddef;
176
177 vardef get_strokep(expr idx) =
178   obstackp[find_whatever(otstroke,idx)]
179 enddef;
180
181 vardef get_strokeq(expr idx) =
182   obstackq[find_whatever(otstroke,idx)]
183 enddef;
184
185 vardef pop_hook =
186   obstacktype[find_whatever(othook,0)]:=otnull;
187 enddef;
188
189 vardef pop_lcblob =
190   obstacktype[find_whatever(otlcblob,0)]:=otnull;
191 enddef;
192
193 vardef pop_stroke =
194   obstacktype[find_whatever(otstroke,0)]:=otnull;
195 enddef;
196
197 vardef push_anchor(expr atype,anchor) =
198   obstacktype[sp]:=otanchor;
199   obstackn[sp]:=atype;
200   if pair anchor:
201     obstackt[sp]:=identity shifted anchor;
202   else:
203     obstackt[sp]:=anchor;
204   fi;
205   sp:=sp+1;
206 enddef;
207
208 vardef push_hook(expr stage,htext) =
209   obstacktype[sp]:=othook;

```

```

210  obstackn[sp]:=stage;
211  obstacks[sp]:=htext;
212  sp:=sp+1;
213  endif;
214
215  vardef push_lcblob(expr blob) =
216    obstacktype[sp]:=otlcblob;
217    obstackp[sp]:=blob;
218    sp:=sp+1;
219  endif;
220
221  vardef push_pbox(expr pbnam) =
222    obstacktype[sp]:=otpbox;
223    obstackt[sp]:=identity scaled 900 shifted (50,50);
224    obstacks[sp]:=pbnam;
225    sp:=sp+1;
226  endif;
227
228  vardef push_pbox_explicit(expr pbnam,pbox) =
229    obstacktype[sp]:=otpbox;
230    obstackt[sp]:=pbox;
231    obstacks[sp]:=pbnam;
232    sp:=sp+1;
233  endif;
234
235  vardef push_pbox_toexpand(expr pbnam) =
236    obstacktype[sp]:=otpbox;
237    obstackt[sp]:=identity scaled 1000 shifted (0,100);
238    obstacks[sp]:=pbnam;
239    obstackba.botoexpand[sp]:=true;
240    % message "to expand " & decimal sp;
241    sp:=sp+1;
242    perl_structure:=perl_structure&"["&pbnam&";";
243  endif;
244
245  vardef push_stroke(expr p,q) =
246    obstacktype[sp]:=otstroke;
247    obstackp[sp]:=p;
248    obstackq[sp]:=q;
249    obstackna.bosize[sp]:=100;
250    sp:=sp+1;
251    perl_structure:=perl_structure&"push_stroke";
252  endif;
253
254  vardef replace_lcblob(expr idx)(text blob) =
255    begingroup
256      save oldblob;
257      path oldblob;

```

```

258     oldblob:=obstackp[find_whatever(otlcblob,idx)];
259     obstackp[find_whatever(otlcblob,idx)]:=blob;
260 endgroup;
261 enddef;
262
263 vardef replace_strokep(expr idx)(text curves) =
264     begingroup
265         save oldp;
266         path oldp;
267         oldp:=obstackp[find_whatever(otstroke,idx)];
268         obstackp[find_whatever(otstroke,idx)]:=curves;
269     endgroup;
270     perl_structure:=perl_structure&"replace_strokep";
271 enddef;
272
273 vardef replace_strokeq(expr idx)(text curves) =
274     begingroup
275         save oldq;
276         path oldq;
277         oldq:=obstackq[find_whatever(otstroke,idx)];
278         obstackq[find_whatever(otstroke,idx)]:=curves;
279     endgroup;
280 enddef;
281
282 vardef set_boalternate(expr idx) =
283     obstackba.boalternate[find_stroke(idx)]:=true;
284 enddef;
285
286 vardef set_bokeepshape(expr idx) =
287     obstackba.bokeepshape[find_whatever(otlcblob,idx)]:=true;
288 enddef;
289
290 vardef set_boserif(expr idx,t,srf) =
291     obstacknaa.boserif[find_stroke(idx)][t]:=srf;
292 enddef;
293
294 vardef set_bosize(expr idx,bos) =
295     obstackna.bosize[find_stroke(idx)]:=bos;
296     if bos=0:
297         perl_structure:=perl_structure&"bosize0";
298     fi;
299 enddef;
300
301 vardef set_botip(expr idx,t,bt) =
302     obstacknaa.botip[find_stroke(idx)][t]:=bt;
303 enddef;

```

## frac-intro.mp

```
1 %
2 % Common code for Tsukurimashou fractions
3 % Copyright (C) 2011 Matthew Skala
4 %
5-29 [Standard copyright notice]
30
31 inclusion_lock(fracintro);
32
33 

---


34
35 transform nxf[];
36
37 frac.in.x1=200;
38 frac.in.x2=800;
39 frac.in.y1=latin_wide_baseline;
40 frac.in.y2=latin_wide_top;
41
42 frac.one.y1=0.02[frac.in.y1,frac.in.y2];
43 frac.one.y2=0.40[frac.in.y1,frac.in.y2];
44 frac.one.y3=0.51[frac.in.y1,frac.in.y2];
45 frac.one.y4=0.60[frac.in.y1,frac.in.y2];
46 frac.one.y5=0.98[frac.in.y1,frac.in.y2];
47
48 (frac.one.x1+frac.one.x2)/2=500;
49 frac.one.x2-frac.one.x1=320;
50
51 frac.two.y1=0.04[frac.in.y1,frac.in.y2];
52 frac.two.y2=0.38[frac.in.y1,frac.in.y2];
53 frac.two.y3=0.51[frac.in.y1,frac.in.y2];
54 frac.two.y4=0.62[frac.in.y1,frac.in.y2];
55 frac.two.y5=0.96[frac.in.y1,frac.in.y2];
56
57 (frac.two.x1+frac.two.x3)/2=500;
58 (frac.two.x3-frac.two.x2)=
59   (frac.two.x2-frac.two.x1);
60 frac.two.x3-frac.two.x1=600;
61
62 frac.three.y1=0.06[frac.in.y1,frac.in.y2];
63 frac.three.y2=0.36[frac.in.y1,frac.in.y2];
64 frac.three.y3=0.51[frac.in.y1,frac.in.y2];
65 frac.three.y4=0.64[frac.in.y1,frac.in.y2];
66 frac.three.y5=0.94[frac.in.y1,frac.in.y2];
67
68 (frac.three.x1+frac.three.x4)/2=500;
69 (frac.three.x4-frac.three.x3)=
70   (frac.three.x3-frac.three.x2)=
```

**FRAC**

## FRAC

```

71 (frac.three.x2-frac.three.x1);
72 frac.three.x4-frac.three.x1=700;
73
74 frac.four.y1=0.08[frac.in.y1,frac.in.y2];
75 frac.four.y2=0.34[frac.in.y1,frac.in.y2];
76 frac.four.y3=0.51[frac.in.y1,frac.in.y2];
77 frac.four.y4=0.66[frac.in.y1,frac.in.y2];
78 frac.four.y5=0.92[frac.in.y1,frac.in.y2];
79
80 (frac.four.x1+frac.four.x5)/2=500;
81 (frac.four.x5-frac.four.x4)=
82 (frac.four.x4-frac.four.x3)=
83 (frac.four.x3-frac.four.x2)=
84 (frac.four.x2-frac.four.x1);
85 frac.four.x5-frac.four.x1=800;
86
87 frac.half.y1=0.10*latin_vcentre;
88 frac.half.y2=0.82*latin_vcentre;
89 frac.half.y3=latin_vcentre;
90 frac.half.y4=1.18*latin_vcentre;
91 frac.half.y5=1.90*latin_vcentre;
92
93 (frac.half.x1+frac.half.x2)/2=250;
94 frac.half.x2-frac.half.x1=330;
95
96 vardef hexdig(expr d) =
97   if d<10: decimal d else: char (d+87) fi
98 enddef;
99
100 vardef make__digit__set(expr xfm,thispage,place) =
101   numeric ccount;
102   ccount:=0;
103   forsuffices i=zero,one,two,three,four,five,six,seven,eight,nine:
104     begintuglyph("uFF" & thispage & place & hexdig(ccount),
105       hex(place & hexdig(ccount)));
106     tsu_xform(xfm)(numeral.i);
107     tsu_render;
108   endtsuglyph;
109   ccount:=ccount+1;
110   endfor;
111 enddef;

```

## latin-intro.mp

```
1 %
2 % Shared code for Tsukurimashou latin
3 % Copyright (C) 2011, 2012 Matthew Skala
4 %
5-29 [Standard copyright notice]
30
31 inclusion_lock(latinintro);
32
33 

---


34
35 latin_wide_low_h:=latin_wide_baseline+mbrush_height*0.8;
36 latin_wide_high_h:=latin_wide_top-mbrush_height*0.8;
37 latin_wide_low_r:=latin_wide_baseline+mbrush_height*0.8-7;
38 latin_wide_high_r:=latin_wide_top-mbrush_height*0.8+15;
39 if sharp_corners:
40   latin_wide_low_v:=latin_wide_baseline;
41   latin_wide_high_v:=latin_wide_top;
42 else:
43   latin_wide_low_v:=latin_wide_baseline+mbrush_height*0.8;
44   latin_wide_high_v:=latin_wide_top-mbrush_height*0.8;
45 fi;
46
47 vardef vmetric(expr a) =
48   (a[latin_wide_low_h,latin_wide_high_h])
49 enddef;
50
51 latin_wide_xheight:=vmetric(0.65);
52 latin_wide_xheight_h:=latin_wide_xheight-mbrush_height*0.6;
53 latin_wide_xheight_r:=latin_wide_xheight-mbrush_height*0.6+15;
54 if sharp_corners:
55   latin_wide_xheight_v:=latin_wide_xheight;
56 else:
57   latin_wide_xheight_v:=latin_wide_xheight-mbrush_height*0.5;
58 fi;
59
60 latin_wide_desc:=vmetric(-0.35);
61 latin_wide_desc_h:=latin_wide_desc+mbrush_height*0.6;
62 latin_wide_desc_r:=latin_wide_desc+mbrush_height*0.6-10;
63 if sharp_corners:
64   latin_wide_desc_v:=latin_wide_desc;
65 else:
66   latin_wide_desc_v:=latin_wide_desc+mbrush_height*0.5;
67 fi;
68
69 latin_wide_lc_baselif:=vmetric(0.02);
70
```

```

71
72
73 transform tsu_xf.accentedcap,tsu_xf.cap_upper_accent,tsu_xf.low_centre_accent;
74
75 if is_proportional:
76   tsu_xf.accentedcap=identity;
77 else:
78   xpart tsu_xf.accentedcap=1;
79   xypart tsu_xf.accentedcap=yxpart tsu_xf.accentedcap=0;
80   (500,vmetric(0)) transformed tsu_xf.accentedcap=(500,vmetric(0));
81   (500,vmetric(1)) transformed tsu_xf.accentedcap=(500,vmetric(0.82));
82 fi;
83
84 accent_default[anc_upper]=identity shifted (500,vmetric(0.75));
85 accent_default[anc_grave]=identity
86   shifted (500+0.4*tsu_punct_size,vmetric(0.75));
87 accent_default[anc_acute]=identity
88   shifted (500-0.4*tsu_punct_size,vmetric(0.75));
89 accent_default[anc_wide]=identity xscaled 0.75 shifted (500,vmetric(0.75));
90 accent_default[anc_tilde]=identity xscaled 0.75 shifted (500,vmetric(0.75));
91 accent_default[anc_ring]=identity shifted (500,vmetric(0.75));
92 accent_default[anc_caron_comma]=identity shifted (730,vmetric(0.93));
93 accent_default[anc_lower]=identity shifted (500,vmetric(-0.26));
94 accent_default[anc_lower_connect]=identity shifted (500,vmetric(0));
95 accent_default[anc_centre]=identity
96   scaled ((latin_wide_high_r-latin_wide_low_r)/200) shifted centre_pt;
97
98 % this one is for capitals that have NOT been shrunk
99 tsu_xf.cap_upper_accent=identity shifted (500,vmetric(1.10));
100
101 tsu_xf.low_centre_accent=identity
102   scaled ((latin_wide_xheight_r-latin_wide_low_r)/200)
103   shifted (xpart centre_pt,(latin_wide_xheight_r+latin_wide_low_r)/2);
104
105 vardef tsu_accent.shift_anchors(text c)(expr s) =
106   begingroup;
107   save killflag;
108   boolean killflag;
109   for i:=1 upto max_accent_seen:
110     if unknown accent_has_default[i]:
111       killflag:=true;
112     elseif not accent_has_default[i]:
113       killflag:=true;
114     else:
115       killflag:=false;
116       for j=sp-1 downto 1:
117         if obstacktype[j]=otanchor:
118           if obstackn[j]=i:

```

```

119         killflag:=true;
120     fi;
121     fi;
122     exitif killflag;
123     endfor;
124 fi;
125 if not killflag:
126     def ai = i enddef;
127     def olda = ((0,0) transformed accent_default[i]) enddef;
128     if c:
129         push_anchor(i,accent_default[i]);
130     fi;
131 fi;
132 endfor;
133 endgroup;
134 for i:=sp-1 downto 1:
135     if obstacktype[i]=otanchor:
136         begingroup
137             def ai = obstackn[i] enddef;
138             def olda = ((0,0) transformed obstackt[i]) enddef;
139             if c:
140                 obstackt[i]:=obstackt[i] shifted s;
141             fi;
142         endgroup;
143     fi;
144 endfor;
145 enddef;
146
147
148
149 vardef tsu_accent.up_default_anchors =
150     tsu_default_anchor(anc_upper,accent_default[anc_upper]
151         shifted (0,vmetric(1.10)-vmetric(0.75)));
152     tsu_default_anchor(anc_grave,accent_default[anc_grave]
153         shifted (0,vmetric(1.10)-vmetric(0.75)));
154     tsu_default_anchor(anc_acute,accent_default[anc_acute]
155         shifted (0,vmetric(1.10)-vmetric(0.75)));
156     tsu_default_anchor(anc_wide,identity xscaled 1.2
157         transformed accent_default[anc_wide]
158         shifted (0,vmetric(1.10)-vmetric(0.75)));
159     tsu_default_anchor(anc_tilde,accent_default[anc_tilde]
160         shifted (0,vmetric(1.10)-vmetric(0.75)));
161     tsu_default_anchor(anc_ring,accent_default[anc_ring]
162         shifted (0,vmetric(1.10)-vmetric(0.75)));
163     tsu_default_anchor(anc_caron_comma,
164         accent_default[anc_caron_comma] shifted (200,0));
165     tsu_default_anchor(anc_lower,accent_default[anc_lower]);
166     tsu_default_anchor(anc_lower_connect,accent_default[anc_lower_connect]);

```

```

167   tsu_default_anchor(anc_centre,accent_default[anc_centre]);
168 enddef;
169
170 vardef tsu_accent.low_default_anchors =
171   tsu_default_anchor(anc_upper,accent_default[anc_upper]);
172   tsu_default_anchor(anc_grave,accent_default[anc_grave]);
173   tsu_default_anchor(anc_acute,accent_default[anc_acute]);
174   tsu_default_anchor(anc_wide,accent_default[anc_wide]);
175   tsu_default_anchor(anc_tilde,accent_default[anc_tilde]);
176   tsu_default_anchor(anc_ring,accent_default[anc_ring]);
177   tsu_default_anchor(anc_caron_comma,accent_default[anc_caron_comma]);
178   tsu_default_anchor(anc_lower,accent_default[anc_lower]);
179   tsu_default_anchor(anc_lower_connect,accent_default[anc_lower_connect]);
180   tsu_default_anchor(anc_centre,tsu_xf.low_centre_accent);
181 enddef;
182
183 vardef tsu_accent.clear_default_anchors =
184   tsu_default_anchor(anc_upper,0);
185   tsu_default_anchor(anc_grave,0);
186   tsu_default_anchor(anc_acute,0);
187   tsu_default_anchor(anc_wide,0);
188   tsu_default_anchor(anc_tilde,0);
189   tsu_default_anchor(anc_ring,0);
190   tsu_default_anchor(anc_caron_comma,0);
191   tsu_default_anchor(anc_lower,0);
192   tsu_default_anchor(anc_lower_connect,0);
193   tsu_default_anchor(anc_centre,0);
194 enddef;

```

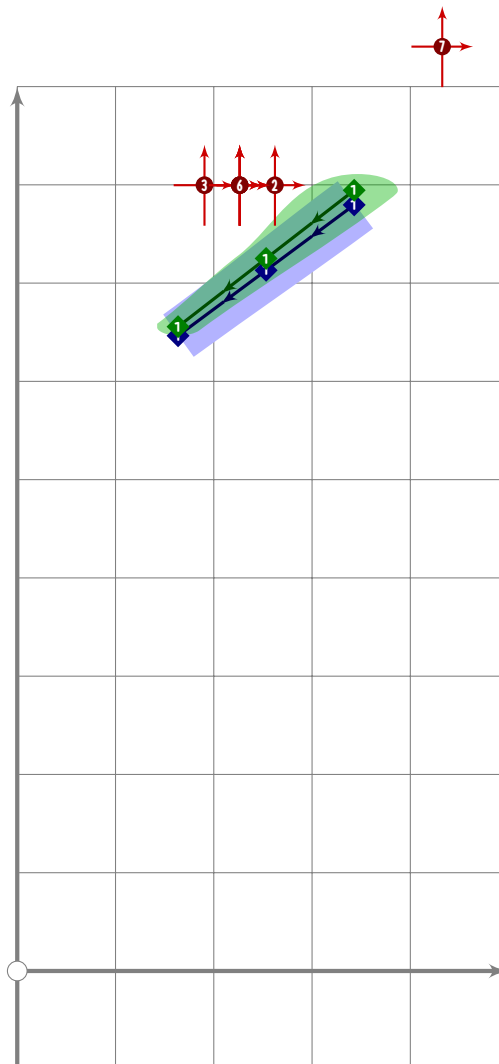
# accent.mp

```

1 %
2 % Accents for Tsukurimashou
3 % Copyright (C) 2011, 2012 Matthew Skala
4 %
5-29 [Standard copyright notice]
30
31 inclusion_lock(accent);
32
33

```

---



ACCE

```

34
35 vardef tsu_accent.acute =
36   push_anchor(-anc_acute,accent_default[anc_acute]);
37   push_stroke(
38     (500+1.1*tsu_punct_size,vmetric(0.95))-
39     (500-0.9*tsu_punct_size,vmetric(0.78)),
40     (2,2)-(1.6,1.6)-(1.3,1.3));
41   replace_strokep(0)(insert_nodes(oldp)(0.5));

```

```

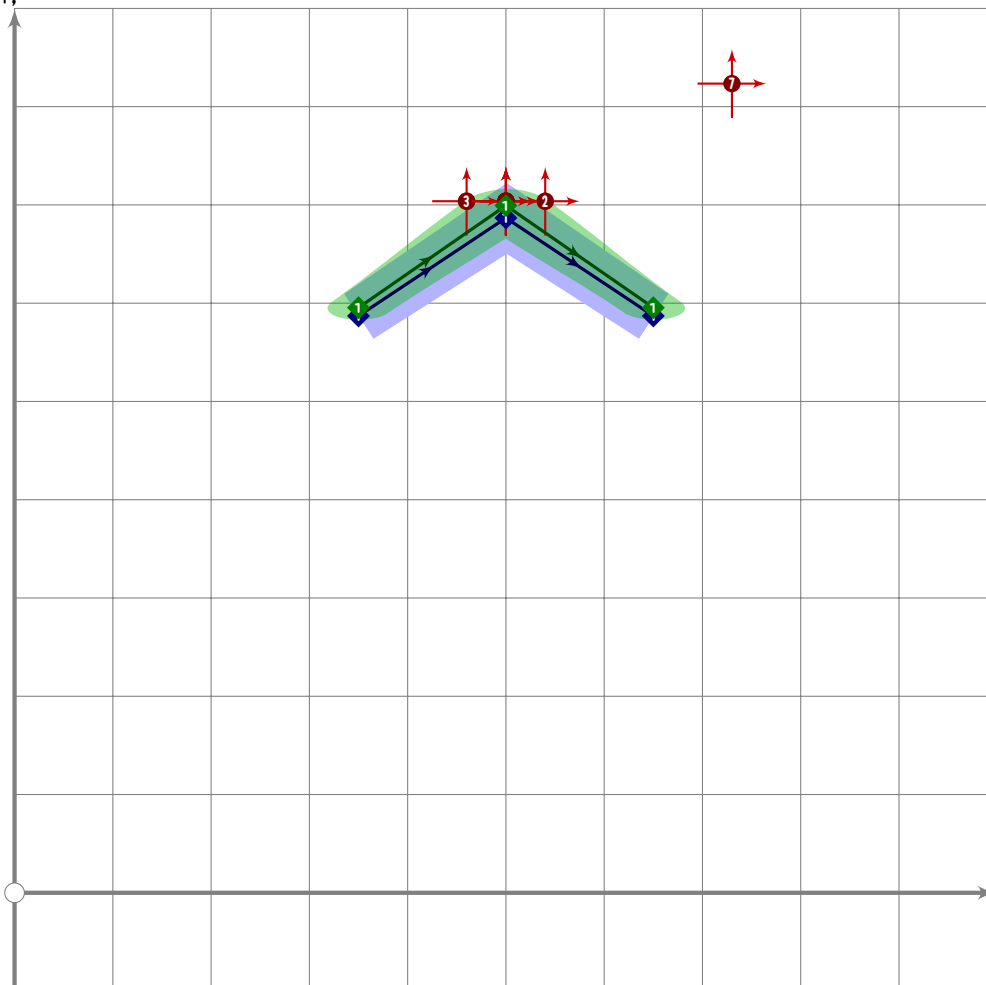
42  set_bosize(0,80);
43  set_botip(0,1,1);
44  push_anchor(anc_upper,accent_default[anc_upper] shifted (-20,150));
45  push_anchor(anc_grave,accent_default[anc_grave] shifted (-20,150));
46  push_anchor(anc_acute,accent_default[anc_acute] shifted (-20,150));
47  push_anchor(anc_wide,accent_default[anc_wide] shifted (-20,150));
48  push_anchor(anc_tilde,accent_default[anc_tilde] shifted (-20,150));
49  push_anchor(anc_ring,accent_default[anc_ring] shifted (-20,150));
50  push_anchor(anc_caron_comma,
51      accent_default[anc_caron_comma] shifted (-20,150));
52 endif;
53
54 vardef tsu_accent.breve =
55     push_anchor(-anc_wide,accent_default[anc_wide]);
56     push_stroke((500-1.3*tsu_punct_size,vmetric(0.95)){down}..
57         (500,vmetric(0.82))..
58         {up}(500+1.3*tsu_punct_size,vmetric(0.95)),
59         (1,1)-(1.9,1.9)-(1,1));
60     push_anchor(anc_upper,accent_default[anc_upper] shifted (0,150));
61     push_anchor(anc_grave,accent_default[anc_grave] shifted (0,150));
62     push_anchor(anc_acute,accent_default[anc_acute] shifted (0,150));
63     push_anchor(anc_wide,accent_default[anc_wide] shifted (0,150));
64     push_anchor(anc_tilde,accent_default[anc_tilde] shifted (0,150));
65     push_anchor(anc_ring,accent_default[anc_ring] shifted (0,150));
66     push_anchor(anc_caron_comma,
67         accent_default[anc_caron_comma] shifted (0,150));
68 endif;
69
70 vardef tsu_accent.caron =
71     push_anchor(-anc_wide,accent_default[anc_wide]);
72     push_stroke((500-1.5*tsu_punct_size,vmetric(0.95))-
73         (500,vmetric(0.80))-
74         (500+1.5*tsu_punct_size,vmetric(0.95)),
75         (2,2)-(1.4,1.4)-(1.4,1.4));
76     set_bosize(0,80);
77     set_botip(0,1,1);
78     push_anchor(anc_upper,accent_default[anc_upper] shifted (0,150));
79     push_anchor(anc_grave,accent_default[anc_grave] shifted (0,150));
80     push_anchor(anc_acute,accent_default[anc_acute] shifted (0,150));
81     push_anchor(anc_wide,accent_default[anc_wide] shifted (0,150));
82     push_anchor(anc_tilde,accent_default[anc_tilde] shifted (0,150));
83     push_anchor(anc_ring,accent_default[anc_ring] shifted (0,150));
84     push_anchor(anc_caron_comma,
85         accent_default[anc_caron_comma] shifted (0,150));
86 endif;
87
88 vardef tsu_accent.caron_comma =
89     push_anchor(-anc_caron_comma,accent_default[anc_caron_comma]);

```

```

90 punct.make_comma((0,0) transformed accent_default[anc_caron_comma],0);
91 enddef;
92
93 vardef tsu_accent.cedilla =
94   push_anchor(-anc_lower_connect,accent_default[anc_lower_connect]);
95   push_stroke(
96     ((0,0)-(-0.3,-1.8){curl 0.7}..(2.6,-2.5)..{curl 0.2}(-2.5,-3.0))
97     scaled (0.5*tsu_punct_size) shifted (500,latin_wide_low_r),
98     (14,14)-(14,14)-(17,17)-(13,13));
99   set_bosize(0,80);
100   set_botip(0,1,1);
101 enddef;

```



ACCE

```

102
103 vardef tsu_accent.circumflex =
104   push_anchor(-anc_wide,accent_default[anc_wide]);
105   push_stroke((500-1.5*tsu_punct_size,ymetric(0.80))-
106     (500,ymetric(0.95))-
107     (500+1.5*tsu_punct_size,ymetric(0.80)),
108     (1.6,1.6)-(2,2)-(1.6,1.6));
109   set_bosize(0,80);
110   set_botip(0,1,1);

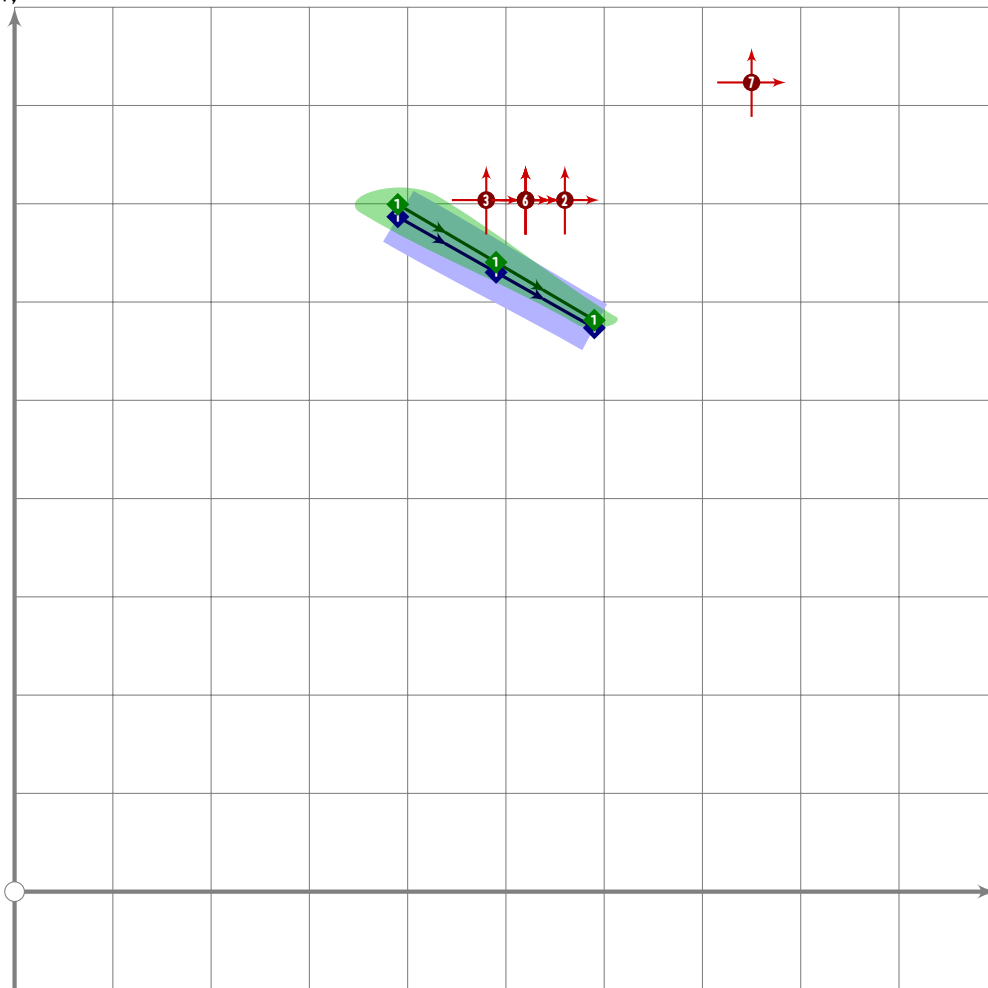
```

```

111 push_anchor(anc_upper,accent_default[anc_upper] shifted (0,150));
112 push_anchor(anc_grave,accent_default[anc_grave] shifted (0,150));
113 push_anchor(anc_acute,accent_default[anc_acute] shifted (0,150));
114 push_anchor(anc_wide,accent_default[anc_wide] shifted (0,150));
115 push_anchor(anc_tilde,accent_default[anc_tilde] shifted (0,150));
116 push_anchor(anc_ring,accent_default[anc_ring] shifted (0,150));
117 push_anchor(anc_caron_comma,
118             accent_default[anc_caron_comma] shifted (0,150));
119 enddef;
120
121 vardef tsu_accent.commabelow =
122   push_anchor(-anc_lower,accent_default[anc_lower]);
123   punct.make_comma((0,0) transformed accent_default[anc_lower],0);
124 enddef;
125
126 vardef tsu_accent.dotabove =
127   push_anchor(-anc_upper,accent_default[anc_upper]);
128   push_lcblob(fullcircle rotated 45 scaled (mbrush_width*1.72+50)
129     shifted ((500,vmetric(0.88))
130       transformed tsu_rescale_xform)
131     transformed inverse tsu_rescale_xform);
132   set_bokeepshape(0);
133   push_anchor(anc_upper,accent_default[anc_upper] shifted (0,150));
134   push_anchor(anc_grave,accent_default[anc_grave] shifted (0,150));
135   push_anchor(anc_acute,accent_default[anc_acute] shifted (0,150));
136   push_anchor(anc_wide,accent_default[anc_wide] shifted (0,150));
137   push_anchor(anc_tilde,accent_default[anc_tilde] shifted (0,150));
138   push_anchor(anc_ring,accent_default[anc_ring] shifted (0,150));
139   push_anchor(anc_caron_comma,
140             accent_default[anc_caron_comma] shifted (0,150));
141 enddef;
142
143 vardef tsu_accent.dotbelow =
144   push_anchor(-anc_lower,accent_default[anc_lower]);
145   push_lcblob(fullcircle rotated 45 scaled (mbrush_width*1.72+50)
146     shifted ((0,0) transformed accent_default[anc_lower]
147       transformed tsu_rescale_xform)
148     transformed inverse tsu_rescale_xform);
149   set_bokeepshape(0);
150 enddef;
151
152 vardef tsu_accent.dotcentred =
153   push_anchor(-anc_centre,accent_default[anc_centre]);
154   push_lcblob(fullcircle rotated 45 scaled (mbrush_width*1.72+50)
155     shifted ((0,0) transformed accent_default[anc_centre]
156       transformed tsu_rescale_xform)
157     transformed inverse tsu_rescale_xform);
158   set_bokeepshape(0);

```

159 endif;



ACCE

160

```

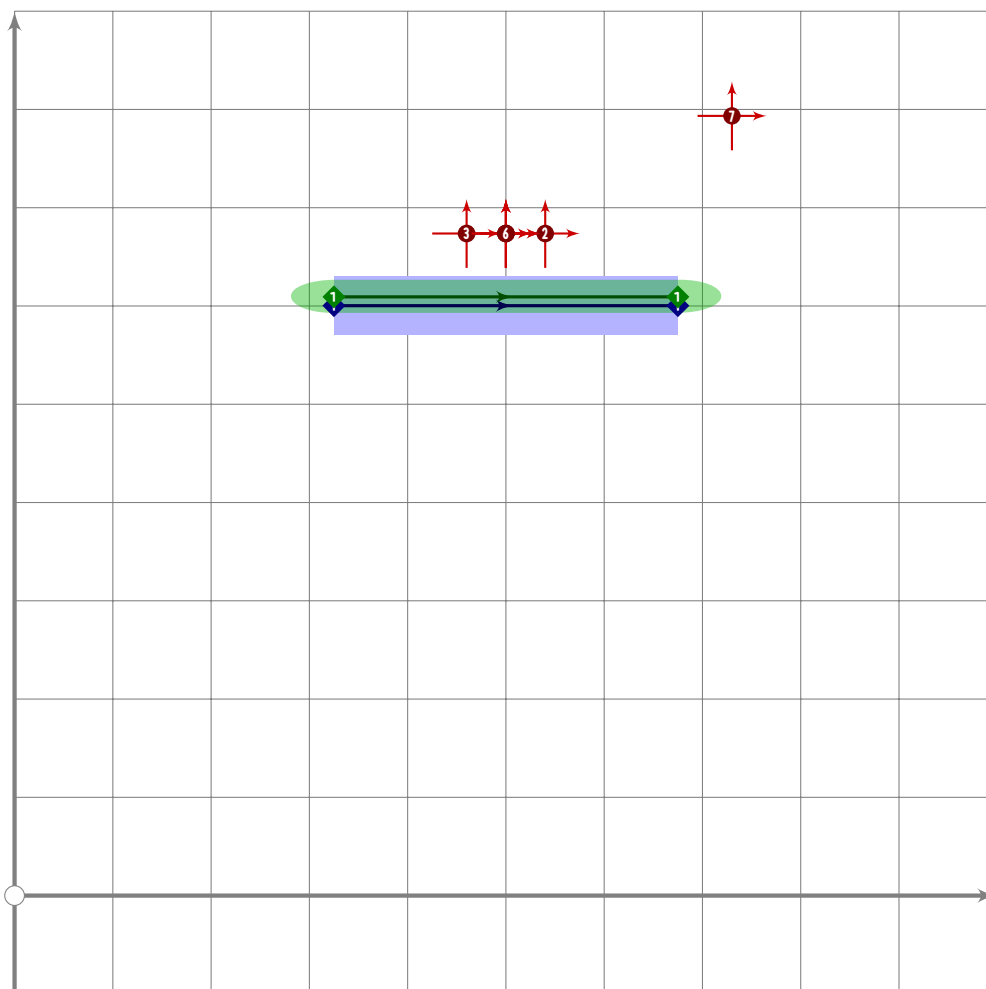
161 vardef tsu_accent.grave =
162   push_anchor(-anc_grave,accent_default[anc_grave]);
163   push_stroke((500-1.1*tsu_punct_size,vmetric(0.95))-
164     (500+0.9*tsu_punct_size,vmetric(0.78)),
165     (2,2)-(1.6,1.6)-(1.3,1.3));
166   replace_strokep(0)(insert_nodes(olddp)(0.5));
167   set_bosize(0,80);
168   set_botip(0,1,1);
169   push_anchor(anc_upper,accent_default[anc_upper] shifted (20,150));
170   push_anchor(anc_grave,accent_default[anc_grave] shifted (20,150));
171   push_anchor(anc_acute,accent_default[anc_acute] shifted (20,150));
172   push_anchor(anc_wide,accent_default[anc_wide] shifted (20,150));
173   push_anchor(anc_tilde,accent_default[anc_tilde] shifted (20,150));
174   push_anchor(anc_ring,accent_default[anc_ring] shifted (20,150));
175   push_anchor(anc_caron_comma,
176     accent_default[anc_caron_comma] shifted (20,150));
177 endif;
178
179 vardef tsu_accent.heavy_metal_umlaut =

```

```

180 push_anchor(-anc_wide,accent_default[anc_wide]);
181 push_lcblob((up-left-down-right-cycle) scaled (mbrush_width*1.5+30)
182   shifted ((500-1.5*tsu_punct_size,vmetric(0.88))
183     transformed tsu_rescale_xform)
184   transformed inverse tsu_rescale_xform);
185 push_lcblob((up-left-down-right-cycle) scaled (mbrush_width*1.5+30)
186   shifted ((500+1.5*tsu_punct_size,vmetric(0.88))
187     transformed tsu_rescale_xform)
188   transformed inverse tsu_rescale_xform);
189 set_bokeepshape(-1);
190 set_bokeepshape(0);
191 push_anchor(anc_upper,accent_default[anc_upper] shifted (0,220));
192 push_anchor(anc_grave,accent_default[anc_grave] shifted (0,220));
193 push_anchor(anc_acute,accent_default[anc_acute] shifted (0,220));
194 push_anchor(anc_wide,accent_default[anc_wide] shifted (0,220));
195 push_anchor(anc_tilde,accent_default[anc_tilde] shifted (0,220));
196 push_anchor(anc_ring,accent_default[anc_ring] shifted (0,220));
197 push_anchor(anc_caron_comma,
198   accent_default[anc_caron_comma] shifted (0,220));
199 enddef;
200
201 vardef tsu_accent.hungarian_umlaut =
202   push_anchor(-anc_wide,accent_default[anc_wide]);
203   push_stroke((500+1.7*tsu_punct_size,vmetric(0.95))-
204     (500+0.7*tsu_punct_size,vmetric(0.78)),
205     (2,2)-(1.6,1.6)-(1.3,1.3));
206   replace_strokep(0)(insert_nodes(oldp)(0.5));
207   set_bosize(0,80);
208   set_botip(0,1,1);
209
210   push_stroke((500-0.4*tsu_punct_size,vmetric(0.95))-
211     (500-1.4*tsu_punct_size,vmetric(0.78)),
212     (2,2)-(1.6,1.6)-(1.3,1.3));
213   replace_strokep(0)(insert_nodes(oldp)(0.5));
214   set_bosize(0,80);
215   set_botip(0,1,1);
216   push_anchor(anc_upper,accent_default[anc_upper] shifted (0,180));
217   push_anchor(anc_grave,accent_default[anc_grave] shifted (0,180));
218   push_anchor(anc_acute,accent_default[anc_acute] shifted (0,180));
219   push_anchor(anc_wide,accent_default[anc_wide] shifted (0,180));
220   push_anchor(anc_tilde,accent_default[anc_tilde] shifted (0,180));
221   push_anchor(anc_ring,accent_default[anc_ring] shifted (0,180));
222   push_anchor(anc_caron_comma,
223     accent_default[anc_caron_comma] shifted (0,180));
224 enddef;

```



ACCE

```

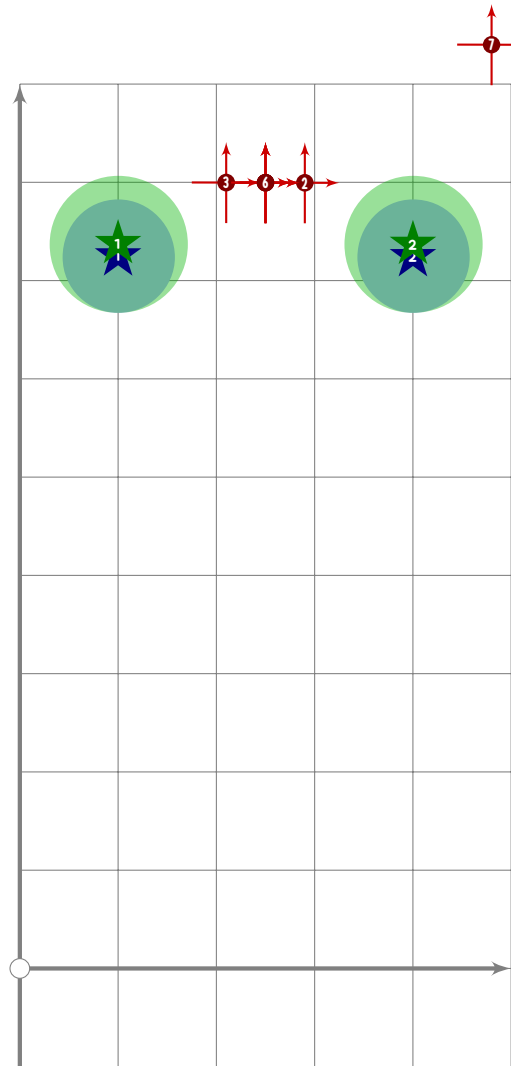
225
226 vardef tsu_accent.macron =
227   push_anchor(-anc_wide,accent_default[anc_wide]);
228   push_stroke((500-1.75*tsu_punct_size,vmetric(0.82))-
229     (500+1.75*tsu_punct_size,vmetric(0.82)),
230     (2,2)-(2,2));
231   set_bosize(0,80);
232   push_anchor(anc_upper,accent_default[anc_upper] shifted (0,120));
233   push_anchor(anc_grave,accent_default[anc_grave] shifted (0,120));
234   push_anchor(anc_acute,accent_default[anc_acute] shifted (0,120));
235   push_anchor(anc_wide,accent_default[anc_wide] shifted (0,120));
236   push_anchor(anc_tilde,accent_default[anc_tilde] shifted (0,120));
237   push_anchor(anc_ring,accent_default[anc_ring] shifted (0,120));
238   push_anchor(anc_caron_comma,
239     accent_default[anc_caron_comma] shifted (0,120));
240 enddef;
241
242 vardef tsu_accent.ringabove =
243   push_anchor(-anc_ring,accent_default[anc_ring]);
244   push_lcblob(fullcircle rotated 45
245     scaled (2*tsu_punct_size+20*tsu_brush_max)

```

```

246     shifted ((500,vmetric(0.83-0.03*mincho)-10)
247         transformed tsu_rescale_xform)
248     transformed inverse tsu_rescale_xform);
249 set_bokeepshape(0);
250 if 2*tsu_punct_size-110*tsu_brush_max>10:
251     push_lcblob(reverse fullcircle rotated 45
252         scaled (2*tsu_punct_size-110*tsu_brush_max)
253         shifted ((500,vmetric(0.83-0.03*mincho)-10)
254             transformed tsu_rescale_xform)
255         transformed inverse tsu_rescale_xform);
256     set_bokeepshape(0);
257 fi;
258 push_anchor(anc_upper,accent_default[anc_upper] shifted (0,170));
259 push_anchor(anc_grave,accent_default[anc_grave] shifted (0,170));
260 push_anchor(anc_acute,accent_default[anc_acute] shifted (0,170));
261 push_anchor(anc_wide,accent_default[anc_wide] shifted (0,170));
262 push_anchor(anc_tilde,accent_default[anc_tilde] shifted (0,170));
263 push_anchor(anc_ring,accent_default[anc_ring] shifted (0,170));
264 push_anchor(anc_caron_comma,
265     accent_default[anc_caron_comma] shifted (0,170));
266 enddef;
267
268 vardef tsu_accent.slash =
269     push_anchor(-anc_centre,accent_default[anc_centre]);
270     push_stroke((( -100,130)-(100,130))
271         transformed accent_default[anc_centre],(2,2)-(2,2));
272     set_bosize(0,86);
273 enddef;
274
275 vardef tsu_accent.tilde =
276     push_anchor(-anc_tilde,accent_default[anc_tilde]);
277     push_stroke(
278         ((-3.5,-0.5){curl 0}{-14,1}..(0,0)..(14,-1)..{curl 0}{3.5,0.5})
279         rotated 5 xyscaled (0.7*tsu_punct_size,0.5*tsu_punct_size)
280         shifted (500,vmetric(0.85)),
281         (0.7,2.7)-(1.7,1.7)-(1.7,1.7)-(1.7,1.7)-(1.7,1.7)-
282         (1.7,1.7)-(0.7,2.7));
283     replace_strokep(0)(insert_nodes(olddp)(0.5,3.5));
284     set_bosize(0,80);
285     push_anchor(anc_upper,accent_default[anc_upper] shifted (0,150));
286     push_anchor(anc_grave,accent_default[anc_grave] shifted (0,150));
287     push_anchor(anc_acute,accent_default[anc_acute] shifted (0,150));
288     push_anchor(anc_wide,accent_default[anc_wide] shifted (0,150));
289     push_anchor(anc_tilde,accent_default[anc_tilde] shifted (0,150));
290     push_anchor(anc_ring,accent_default[anc_ring] shifted (0,150));
291     push_anchor(anc_caron_comma,
292         accent_default[anc_caron_comma] shifted (0,150));
293 enddef;

```



ACCE

```

294
295 vardef tsu_accent.umlaut =
296   push_anchor(-anc_wide,accent_default[anc_wide]);
297   push_lcblob(fullcircle rotated 45 scaled (mbrush_width*1.72+50)
298     shifted ((500-1.5*tsu_punct_size,vmetric(0.88))
299       transformed tsu_rescale_xform)
300     transformed inverse tsu_rescale_xform);
301   push_lcblob(fullcircle rotated 45 scaled (mbrush_width*1.72+50)
302     shifted ((500+1.5*tsu_punct_size,vmetric(0.88))
303       transformed tsu_rescale_xform)
304     transformed inverse tsu_rescale_xform);
305   set_bokeepshape(-1);
306   set_bokeepshape(0);
307   push_anchor(anc_upper,accent_default[anc_upper] shifted (0,150));
308   push_anchor(anc_grave,accent_default[anc_grave] shifted (0,150));
309   push_anchor(anc_acute,accent_default[anc_acute] shifted (0,150));
310   push_anchor(anc_wide,accent_default[anc_wide] shifted (0,150));
311   push_anchor(anc_tilde,accent_default[anc_tilde] shifted (0,150));
312   push_anchor(anc_ring,accent_default[anc_ring] shifted (0,150));

```

```

313   push_anchor(anc_caron_comma,
314               accent_default[anc_caron_comma] shifted (0,150));
315   enddef;
316
317   _____
318
319   vardef tsu_accent.capital(text curves) =
320     tsu_xform(tsu_xf.accentedcap)
321     (curves;tsu_accent.shift_anchors(true)((0,0)));
322   enddef;
323
324   vardef tsu_accent.apply(text basecurves)(text markcurves) =
325     begingroup;
326     save xsp,ysp,bmi,mbi,bmt,killflag;
327     numeric xsp,ysp,bmi,mbi;
328     transform bmt;
329     boolean killflag;
330     basecurves;
331     xsp:=sp;
332     markcurves;
333     killflag:=false;
334     for i:=sp-1 downto xsp:
335       if obstacktype[i]=otanchor:
336         if obstackn[i]<0:
337           mbi:=i;
338           killflag:=true;
339         fi;
340       fi;
341       exitif killflag;
342     endfor;
343     if known mbi:
344       if known accent_default[-obstackn[mbi]]:
345         bmt:=accent_default[-obstackn[mbi]];
346       else:
347         bmt:=accent_default[anchor_parent[-obstackn[mbi]]];
348       fi;
349       killflag:=false;
350       for i:=xsp-1 downto 1:
351         if obstacktype[i]=otanchor:
352           if obstackn[i]=-obstackn[mbi]:
353             bmi:=i;
354             bmt:=obstackt[i];
355             killflag:=true;
356           fi;
357         fi;
358         exitif killflag;
359       endfor;
360       if (not killflag) and (known anchor_parent[-obstackn[mbi]]):

```

```

361     for i:=xsp-1 downto 1:
362         if obstacktype[i]=otanchor:
363             if obstackn[i]=anchor_parent[-obstackn[mbi]]:
364                 bmi:=i;
365                 bmt:=obstackt[i];
366                 killflag:=true;
367             fi;
368         fi;
369         exitif killflag;
370     endfor;
371 fi;
372 obstacktype[mbi]:=otnull;
373 if known bmi:
374     obstacktype[bmi]:=otnull;
375 fi;
376 ysp:=sp;
377 sp:=xsp;
378 tsu_xform((inverse obstackt[mbi]) transformed bmt)(sp:=ysp);
379 fi;
380 endgroup;
381 enddef;

```

## bcircle.mp

```
1 %
2 % Bounding circle algorithm of E. Welzl
3 % Copyright (C) 2011 Matthew Skala
4 %
5-29 [Standard copyright notice]
30
31 inclusion_lock(bcicle);
32
33 % swap points in pts[] array
34 vardef swap_pts(expr a,b)=
35   pair tmppt;
36   tmppt:=pts[a];
37   pts[a]:=pts[b];
38   pts[b]:=tmppt;
39 enddef;
40
41 % compute bounding circle on up to three points
42 vardef bcircle.basis(expr rstart,rend) =
43   if rend<=rstart+1:
44     identity
45   else:
46     begingroup
47       save x,y,myt;
48       numeric x[],y[];
49       transform myt;
50       z1=pts[rstart];
51       z2=pts[rstart+1];
52       xypart myt=0;
53       yxpart myt=0;
54       if rend=rstart+2:
55         z3=(z1+z2)/2;
56         (0,0) transformed myt=(z1+z2)/2;
57         xxpart myt=yy part myt=abs(z1-z3);
58       else:
59         z3=pts[rstart+2];
60         z4=(z1+z2)/2;
61         z5=(z1+z3)/2;
62         z6=z4+whatever*((z2-z1) rotated 90);
63         z6=z5+whatever*((z3-z1) rotated 90);
64         (0,0) transformed myt=z6;
65         xxpart myt=yy part myt=abs(z1-z6);
66       fi;
67       myt
68     endgroup
69   fi
70 enddef;
```

```

71
72 % recursion to compute bounding circle.
73 % Input point sets are in pts[] array, arguments are indices into it
74 vardef bcircle.internal(expr pstart,rstart,rend) =
75   if (pstart=rstart) or (rend-rstart=3):
76     bcircle.basis(rstart,rend)
77   else:
78     begingroup
79       transform d;
80       pind:=floor ((rstart-pstart)*uniformdeviate 1)+pstart;
81       swap_pts(pstart,pind);
82       d=bcircle.internal(pstart+1,rstart,rend);
83       pair xpt;
84       xpt transformed d=pts[pstart];
85       if abs(xpt)>1:
86         swap_pts(pstart,(rstart-1));
87         d:=bcircle.internal(pstart,rstart-1,rend);
88       fi;
89       d
90     endgroup
91   fi
92 enddef;
93
94 % wrapper for bounding circle algorithm - compute bcircle of points
95 vardef bcircle.points(text txt) =
96   begingroup
97     save d,tmppt,pind,xpt,pts,pcnt;
98     pcnt:=0;
99     for myp=txt:
100       pts[pcnt]:=myp;
101       pcnt:=pcnt+1;
102     endfor;
103     bcircle.internal(0,pcnt,pcnt)
104   endgroup
105 enddef;
106
107 % wrapper for bounding circle algorithm - compute bcircle of paths
108 vardef bcircle.paths(text txt) =
109   begingroup
110     save d,tmppt,pind,xpt,pts,pcnt;
111     pcnt:=0;
112     for myp=txt:
113       for i=0 step 0.1 until length myp:
114         pts[pcnt]:=point i of myp;
115         pcnt:=pcnt+1;
116       endfor
117     endfor;
118     bcircle.internal(0,pcnt,pcnt)

```

```
119 endgroup
120 \enddef;
```

## buildkanji.mp

```
1 %
2 % Build a kanji character by assembling parts
3 % Copyright (C) 2011, 2012, 2013 Matthew Skala
4 %
5-29 [Standard copyright notice]
30
31 inclusion_lock(buildkanji);
32
33 

---


34
35 % a beret is a tilted hat
36 vardef build_kanji.add_beret(text curves) =
37   perl_structure:=perl_structure
38   &"['build_kanji.add_beret';eids.u2FF1.u4E3F.1";
39   begingroup
40     save osp;
41     numeric osp;
42     osp:=sp;
43     curves;
44     save i,lox,hix,toppt,myxf;
45     lox:=infinity;
46     hix:=-infinity;
47     pair toppt;
48     toppt:=(500,-infinity);
49     i:=0;
50     forever:
51       exitif find_stroke(i)<osp;
52       if xpart llcorner get_strokep(i)<lox:
53         lox:=xpart llcorner get_strokep(i);
54       fi;
55       if xpart urcorner get_strokep(i)>hix:
56         hix:=xpart urcorner get_strokep(i);
57       fi;
58       if ypart point 0 of get_strokep(i)>ypart toppt:
59         toppt:=point 0 of get_strokep(i);
60       fi;
61       if ypart point infinity of get_strokep(i)>ypart toppt:
62         toppt:=point infinity of get_strokep(i);
63       fi;
64       i:=i+1;
65     endfor;
66     i:=0;
67     forever:
68       exitif find_stroke(i)<osp;
69       if point 0 of get_strokep(i)=toppt:
70         set_boserif(i,0,whatever);
```

**BUIL**

```

71     fi;
72     if point infinity of get_strokep(i)=toppt:
73         set_boserif(i,length get_strokep(i),whatever);
74     fi;
75     i:=i-1;
76 endfor;
77 transform myxf;
78 (-500,810) transformed myxf=(lox,(ypart toppt)+30);
79 (500,900) transformed myxf=(hix,900);
80 (0,780) transformed myxf=toppt;
81 push_stroke(
82     ((-400,750)..(0,780)..tension 11..(360,840)) transformed myxf,
83     (1.1,1.1)-(1.6,1.6)-(2.0,2.0));
84 endgroup;
85 perl_structure:=perl_structure&"],";
86 enddef;
87
88 vardef build_kanji.add_jtail(expr idx) =
89     replace_strokep(idx)(oldp-(xpart point infinity of oldp,30){down}..
90     {curl 0.2}((xpart point infinity of oldp)-150,0));
91     replace_strokep(idx)(insert_nodes(oldp)((length oldp)-0.5));
92     replace_strokeq(idx)(oldq-(1.5,1.5)-(1.4,1.4)-(1.2,1.2));
93 enddef;
94
95 % hook used by extend_ltail_enclose
96 numeric last_ltail;
97
98 vardef build_kanji.add_ltail(expr idx) =
99     begingroup
100         save x,y;
101         numeric x[],y[];
102         z1=point infinity of get_strokep(idx);
103         last_ltail:=find_stroke(idx);
104         x2=x1;
105         y2=80-40*mincho;
106         x3=0.4[x2,800];
107         y3=mincho[0,-20];
108         replace_strokep(idx)(interpath(mincho,
109             oldp-z2{down}..{right}z3..(750,0)..(810,30)..
110             tension 2..(850,230),
111             oldp-z2{down}..{right}z3..(850,0){curl 0.2}..(810,60)..
112             tension 2..(810,170)));
113         replace_strokeq(idx)(oldq-(1.6,1.6)-(1.75,1.75)-(1.9,1.9)
114             -(1.3,1.3)-(0.8,0.8));
115     endgroup;
116 enddef;
117
118 vardef build_kanji.attach_fishhook(expr scaleamt)(text curves) =

```

**BUIL**

```

119 perl_structure:=
120   perl_structure&"['build_kanji.attach_fishhook,'eids.u2FF1.u2E88._2,['";
121 begingroup
122   save osp;
123   numeric osp;
124   osp:=sp;
125   tsu_xform(identity shifted (0,50) yscaled scaleamt shifted (0,-50))
126     (curves);
127   perl_structure:=perl_structure&"]";
128   save i,j,x,y,pp,myxf;
129   path pp;
130   transform myxf;
131   z1=z2=(-infinity,-infinity);
132   i:=0;
133   forever:
134     exitif find_stroke(i)<osp;
135     pp:=get_strokep(i) rotated -45;
136     for j=0 upto length pp:
137       x3:=xpart point j of pp;
138       y3:=ypart point j of pp;
139       if x3>x1:
140         x1:=x3;
141         y1:=y3;
142       fi;
143       if y3>y2:
144         x2:=x3;
145         y2:=y3;
146       fi;
147     endfor;
148     i:=i+1;
149   endfor;
150   (0,0) transformed myxf=(z2 rotated 45);
151   (1,8,0) transformed myxf=(z1 rotated 45);
152   xpart myxf=0;
153   ypart ((0,1) transformed myxf)=830;
154   push_stroke(
155     ((0.5,0.9)..tension 1.2..(0,0)..(-0.5,-0.5)) transformed myxf,
156     (1.7,1.7)-(1.3,1.3)-(1,1));
157   set_boserif(0,0,10);
158   push_stroke(
159     ((0.4,0.65)-(1.3,0.65)..tension 1.2..(1.14,0.3)..(0.88,0))
160     transformed myxf,
161     (1.6,1.6)-(1.6,1.6)-(1.4,1.4)-(1,1));
162   set_boserif(0,1,4);
163   set_botip(0,1,0);
164 endgroup;
165 perl_structure:=perl_structure&"]";
166 enddef;

```

**BUIL**

```

167
168 vardef build_kanji.attach_tick(expr newtop)(text curves) =
169   perl_structure:=
170     perl_structure&"['build_kanji.attach_fishhook';eids.u2FF1.u31D2._1';[";
171   begingroup
172     save atosp,atnsp;
173     numeric atosp,atnsp;
174     atosp:=sp;
175     curves;
176     perl_structure:=perl_structure&"']";
177     atnsp:=sp;
178     save i,thisy,maxy;
179     maxy:=-infinity;
180     i:=0;
181     forever:
182       exitif find_stroke(i)<atosp;
183       if (xpart (get_stroke(i) intersectiontimes
184         ((400,900)-(500,200)-(600,900))))>=0:
185         thisy:=ypart urcorner get_stroke(i);
186         if thisy>maxy:
187           maxy:=thisy;
188         fi;
189       fi;
190       i:=i-1;
191     endfor;
192     sp:=atosp;
193     tsu_xform(identity yscaled (newtop/maxy))(sp:=atnsp);
194     push_stroke((500,190+newtop)-(440,newtop),(1,7,1,7)-(1,1));
195     set_boserif(0,0,10);
196   endgroup;
197   perl_structure:=perl_structure&"']";
198 enddef;
199
200 vardef hook.box_bottom(expr sides_i,bottom_i) =
201   if (obstacktype[sides_i]=otstroke) and (obstacktype[bottom_i]=otstroke):
202     if (3=length obstackp[sides_i]) and (1=length obstackp[bottom_i]):
203       begingroup;
204         save x,y,p,e;
205         numeric x[],y[],e[];
206         path p[];
207
208         p1=obstackp[sides_i];
209         p2=obstackp[bottom_i];
210
211         z1=-(direction 0 of p1)/abs(direction 0 of p1);
212         z2=(direction 0.5 of p2)/abs(direction 0.5 of p2);
213         z3=(direction 3 of p1)/abs(direction 3 of p1);
214

```

```

215         if (abs(z1 dotprod z2)<0.1) and (abs(z3 dotprod z2)<0.1):
216             point 0 of p1=z4+e1*z1;
217             z4=(point 0 of p2)+whatever*z2;
218             point 3 of p1=z5+e2*z3;
219             z5=(point 1 of p2)+whatever*z2;
220             e3=obstackna.bosize[bottom_i]*tsu_brush_max*tsu_brush_shape*0.5;
221
222             if e1<e3:
223                 p1:=(z4+e3*z1)-(subpath (1,3) of p1);
224             fi;
225             if e2<e3:
226                 p1:=(subpath (0,2) of p1)-(z5+e3*z3);
227             fi;
228             obstackp[sides_i]:=p1;
229         fi;
230     endgroup;
231 fi;
232 fi;
233 enddef;
234
235 vardef build_kanji.box(expr ul,lr) =
236     perl_structure:=perl_structure&"['build_kanji.box;";
237     begingroup
238         save boxext;
239         if (ypart (ul-lr))>500:
240             boxext:=-100/(ypart (ul-lr));
241         else:
242             boxext:=-0.2;
243         fi;
244         if (boxext)[ypart lr,ypart ul]<=-60:
245             boxext:=(-60+ypart lr)/(ypart ul-ypart lr);
246         fi;
247         push_stroke((xpart ul,(boxext)[ypart lr,ypart ul])-ul-
248             (xpart lr,ypart ul)-(xpart lr,(boxext)[ypart lr,ypart ul]),
249             (1.5,1.5)-(1.7,1.7)-(1.7,1.7)-(1.5,1.5));
250     endgroup;
251     set_botip(0,1,1);
252     set_botip(0,2,1);
253     set_boserif(0,1,4);
254     set_boserif(0,2,4);
255     push_stroke((xpart ul,ypart lr)-lr,
256         (1.5,1.5)-(1.5,1.5));
257     push_hook(hsmain_render,
258         "hook.box_bottom("&(decimal find_stroke(-1))&";"
259             &(decimal find_stroke(0))&");" );
260     perl_structure:=perl_structure&"]";
261 enddef;
262

```

```

263 vardef build_kanji.cliff_enclose(text contents) =
264   push_pbox_toexpand("build_kanji.cliff_enclose");
265   perl_structure:=perl_structure&"eids.u2FF8.u5382.1_";
266   push_stroke((50,-50)..(120,100)..(160,300)..tension 1.2..(180,760)
267     -(850,760),
268     (1,1)-(1.3,1.3)-(1.5,1.5)-(1.6,1.6)-(1.6,1.6));
269   set_botip(0,3,1);
270   begingroup
271     save t;
272     transform t;
273     (50,-50) transformed t=(230,-50);
274     (500,-50) transformed t=(560,-50);
275     (500,850) transformed t=(560,730);
276     tsu_xform(t)(contents);
277   endgroup;
278   expand_pbox;
279 enddef;
280
281 vardef build_kanji.cup(expr ul,lr) =
282   push_stroke(ul-(xpart ul,(-0.2)[ypart lr,ypart ul]),
283     (1.6,1.6)-(1.4,1.4));
284   set_boserif(0,0,10);
285   push_stroke((xpart lr,ypart ul)-(xpart lr,(-0.2)[ypart lr,ypart ul]),
286     (1.6,1.6)-(1.4,1.4));
287   set_boserif(0,0,10);
288   push_stroke((xpart ul,ypart lr)-lr,
289     (1.5,1.5)-(1.5,1.5));
290   perl_structure:=perl_structure&"build_kanji.cup";
291 enddef;
292
293 vardef build_kanji.dotcliff_enclose(text contents) =
294   push_pbox_toexpand("build_kanji.dotcliff_enclose");
295   perl_structure:=perl_structure&"eids.u2FF8.u5E7F.2_";
296   push_stroke((550,810)-(550,660),
297     (1.6,1.6)-(1.5,1.5));
298   set_boserif(0,0,10);
299   push_stroke(
300     (50,-50)..(120,100)..(160,300)..tension 1.2..(180,660)-(850,660),
301     (1,1)-(1.3,1.3)-(1.5,1.5)-(1.6,1.6)-(1.6,1.6));
302   set_botip(0,3,1);
303   begingroup
304     save t;
305     transform t;
306     (50,-50) transformed t=(230,-50);
307     (500,-50) transformed t=(560,-50);
308     (500,850) transformed t=(560,630);
309     tsu_xform(t)(contents);
310   endgroup;

```

**BUIL**

```

311 expand_pbox;
312 enddef;
313
314 vardef build_kanji.flag_enclose(expr xs,ys)(text contents) =
315   push_pbox_toexpand("build_kanji.dotcliff_enclose");
316   perl_structure:=perl_structure&"eids.u2FF8.u5C38.1_";
317   string flag_enclose.ps;
318   flag_enclose.ps:=perl_structure;
319   build_kanji.lr(480,60)
320     (kanji.grtwo.direction;
321     (perl_structure:=flag_enclose.ps;
322     push_stroke((300,810)..tension 1.2..(220,600)..(110,480),
323       (1.6,1.6)-(1.4,1.4)-(1,1));
324     set_boserif(0,0,10);
325     push_stroke((100,680)-(780,680),(1.6,1.6)-(1.6,1.6));
326     set_boserif(0,1,9);
327     replace_strokep(0)(subpath
328       (0.01*xpart (oldp intersectiontimes get_strokep(-1)),1) of oldp);
329     tsu_xform(identity shifted (-500,0) xyscaled (xs,ys)
330       shifted (500,-20))
331     (contents);
332     flag_enclose.ps:=perl_structure);
333   expand_pbox;
334   perl_structure:=flag_enclose.ps&"1_";
335 enddef;
336
337 vardef build_kanji.gate_enclose(text contents) =
338   perl_structure:=perl_structure&"['build_kanji.gate_enclose";
339   perl_structure:=perl_structure&"eids.u2FF5.u9580.1_";
340   kanji.grtwo.gate;
341   begingroup
342     transform xf;
343     (50,-50) transformed xf=(220,40);
344     (950,850) transformed xf=(780,420);
345     xypart xf=yxpart xf=0;
346     tsu_xform(xf)(contents);
347   endgroup;
348   perl_structure:=perl_structure&"1_";
349 enddef;
350
351 vardef build_kanji.harmonic(expr gap,sval,lspread)(text curves) =
352   perl_structure:=perl_structure
353     &"['build_kanji.harmonic";eids.u2FF1.u003F.1_1_";
354   begingroup
355     save myxf;
356     transform myxf;
357     (50,-50) transformed myxf=(50,-50);
358     (950,50) transformed myxf=(950,50);

```

```

359 (50,850) transformed myxf=(50,850-gap);
360 tsu_xform(myxf)(curves);
361 perl_structure:=perl_structure&"],[";
362 save hsp;
363 hsp:=sp;
364 tsu_xform(myxf)(build_kanji.spread_legs(lspread)(curves));
365 save i,tp,toclip,nadded;
366 numeric i,toclip,nadded;
367 path tp;
368 i:=0;
369 toclip:=0;
370 nadded:=0;
371 forever:
372   tp:=get_strokep(i);
373   if (ypart ulcorner tp<450
374     or (abs(ypart (direction 0 of tp)/abs(direction 0 of tp))>0.95)
375     or (abs(ypart (direction infinity of tp)/
376       abs(direction infinity of tp))>0.95):
377     set_bosize(i,0);
378     toclip:=toclip+1;
379   else:
380     replace_strokep(i)
381       (tp shifted -(0.5[ulcorner tp,lrcorner tp])
382         scaled sval
383         shifted ((0,gap)+0.5[ulcorner tp,lrcorner tp]));
384     set_bosize(i)(get_bosize(i)*sqrt(sval));
385     toclip:=toclip+2;
386     nadded:=nadded+1;
387   fi;
388   i:=i-1;
389   exitif find_stroke(i)<hsp;
390 endfor;
391 endgroup;
392 perl_structure:=perl_structure&""]];
393 enddef;
394
395 vardef build_kanji.lean_to(expr lr) =
396   push_stroke((120,620)-(880,620),(1.6,1.6)-(1.6,1.6));
397   set_bosierif(0,1,9);
398   begingroup
399     save ltxf;
400     transform ltxf;
401     xypart ltxf=yxpart ltxf=0;
402     (60,800) transformed ltxf=(60,800);
403     (360,-30) transformed ltxf=lr;
404     push_stroke(
405       ((360,800)..tension 1.2..(270,300)..(60,-30)) transformed ltxf,
406       (1.6,1.6)-(1.4,1.4)-(0.9,0.9));

```

**BUIL**

```

407 endgroup;
408 set_boserif(0,0,10);
409 enddef;
410
411 vardef build_kanji.level(text curves) =
412   begingroup
413     save xsp;
414     xsp:=sp;
415     curves;
416     save lsum,denom,i;
417     lsum:=0;
418     denom:=0;
419     i:=0;
420     forever:
421       exitif find_stroke(i)<xsp;
422       if unknown get_bosize(i):
423         set_bosize(i,100);
424       fi;
425       if (get_bosize(i)>0):
426         lsum:=lsum+mlog(get_bosize(i));
427         denom:=denom+1;
428       fi;
429       i:=i+1;
430     endfor;
431     i:=0;
432     forever:
433       exitif find_stroke(i)<xsp;
434       if get_bosize(i)>0:
435         set_bosize(i,mexp(lsum/denom));
436       fi;
437       i:=i+1;
438     endfor;
439   endgroup;
440 enddef;
441
442 % note special calling convention - extra boolean for inclusion of "tick"
443 % seen in some Japanese and Korean characters
444 vardef build_kanji.long_stride_enclose(expr do_tick)(text contents) =
445   push_pbox_toexpand("build_kanji.long_stride_enclose");
446   perl_structure:=perl_structure&"eids.u2FFA.u5EF4._3";
447   begingroup
448     save myxf;
449     transform myxf;
450     (50,850) transformed myxf=(350,810);
451     (50,50) transformed myxf=(350,80);
452     (950,50) transformed myxf=(950,80);
453     tsu_xform(myxf)(contents);
454     save x,y;

```

```

455   numeric x[],y[];
456   x1=80;
457   x2=260;
458   x4=120;
459   y1=y2=780;
460   y4=460;
461   z3=0.4[z4,z2]*mincho*(30,0);
462   z5=(0.3-0.5*mincho)[z4,z2];
463   push_stroke(z1-z2..tension 1.2..z3..z5,
464     (1.6,1.6)-(1.6,1.6)-(1.6-0.1*mincho,1.6-0.1*mincho)-
465     (1.6-0.2*mincho,1.6-0.2*mincho));
466   set_boserif(0,1,4);
467   set_botip(0,1,0);
468   x7=0;
469   x9=300;
470   x10=220;
471   x11=100;
472   y7=y9=y4;
473   y10=80;
474   y11=-70;
475   z6=point 2.2 of get_strokep(0);
476   z8=(z7-z9) intersectionpoint (get_strokep(0)-((-1)[z4,z2]));
477   push_stroke(z6-z8-z9..tension 1.2..z10..z11,
478     (1.6-0.5*mincho,1.6-0.5*mincho)-(1.5-0.1*mincho,1.5-0.1*mincho)-
479     (1.6,1.6)-(1.4,1.4)-(1,1));
480   set_boserif(0,2,4);
481   set_botip(0,1,1);
482   set_botip(0,2,0);
483   if do_tick:
484     push_stroke((150,330)..(120,290)..(60,220),
485       (1.2,1.2)-(1.1,1.1)-(1.6,1.6));
486   fi;
487   push_stroke((130,300)..(390,30)..tension 1.6..(900,-50),
488     (1,1)-(1.6,1.6)-(1.9,1.9));
489   endgroup;
490   expand_pbox;
491   enddef;
492
493   vardef build_kanji.lstransform(expr thresh,dist,mypt) =
494     if ypart mypt>thresh:
495       mypt
496     else:
497       (xpart mypt,
498         (ypart mypt)*((thresh-dist)/thresh)
499         +(xpart mypt/1000)[(dist/thresh)*ypart mypt,dist])
500     fi
501   enddef;
502

```

```

503 vardef build_kanji.lift_skirt(expr thresh,dist)(text curves) =
504   begingroup
505     save osp;
506     numeric osp;
507     osp:=sp;
508     curves;
509     save i,boti,x,y;
510     numeric x[],y[];
511     i:=0;
512     boti:=whatever;
513     y0:=1000;
514     forever:
515       exitif find_stroke(i)<osp;
516       if (ypart llcorner get_strokep(i))=(ypart urcorner get_strokep(i)):
517         if (unknown boti) or (ypart llcorner get_strokep(i)<y0):
518           y0:=ypart llcorner get_strokep(i);
519           boti:=i;
520         fi;
521       fi;
522       replace_strokep(i)(
523         for j=0 upto length oldp-1:
524           build_kanji.lstransform(thresh,dist)(point j of oldp)
525             ..controls build_kanji.lstransform(thresh,dist)(postcontrol j of oldp)
526             and build_kanji.lstransform(thresh,dist)(precontrol j+1 of oldp)..
527         endfor
528         if cycle oldp:
529           cycle
530         else:
531           build_kanji.lstransform(thresh,dist)(point infinity of oldp)
532         fi);
533       i:=i-1;
534     endfor;
535     if (known boti) and (y0<=thresh):
536       replace_strokep(boti)
537         (((0,7)+point 0 of oldp)..tension 1.2..((0,-5)+point 0.5 of oldp)..
538         ((0,10)+point 1 of oldp));
539       replace_strokeq(boti)((1.7,1.7)-(1.5,1.5)-(1,1));
540       set_bosserif(boti,1,whatever);
541     fi;
542   endgroup;
543 enddef;
544
545 vardef build_kanji.lr(expr splitpoint,overlap)
546   (text leftstuff)(text rightstuff) =
547   perl_structure:=perl_structure
548   &"['build_kanji.lr",eids.u2ff0._1.1_',[";
549   begingroup
550     save t;

```

```

551   transform t[];
552   yypart t1=yypart t2=1;
553   ypart t1=yxpart t1=xy part t1=y part t2=yxpart t2=xy part t2=0;
554   (50,0) transformed t1=(50,0);
555   (950,0) transformed t1=(splitpoint+overlap/2,0);
556   (50,0) transformed t2=(splitpoint-overlap/2,0);
557   (950,0) transformed t2=(950,0);
558   tsu_xform(t1)(leftstuff);
559   perl_structure:=perl_structure&"],[";
560   tsu_xform(t2)(rightstuff);
561   endgroup;
562   perl_structure:=perl_structure&""]";
563   endif;
564
565   vardef build_kanji.road_enclose(text contents) =
566     push_pbox__toexpand("build_kanji.road_enclose");
567     perl_structure:=perl_structure&"eids.u2FFA.u2ECC._3";
568     begingroup
569       save myxf;
570       transform myxf;
571       (50,850) transformed myxf=(315,850);
572       (50,50) transformed myxf=(315,50);
573       (950,50) transformed myxf=(950,50);
574       tsu_xform(myxf)(contents);
575       push_stroke((100,770)..tension 1.2..(180,690)..(220,630),
576         (1,1)-(1.3,1.3)-(1.9,1.9));
577       set_bosize(0,92);
578       push_stroke((80,453)-(240,450)..(mincho[230,210],250)
579         ..tension 1.2..(mincho[180,140],50)..{curl 0.4}(60,-40),
580         (1,1)-(1.6,1.6)-(1,1)-(1.2,1.2)-(1,1));
581       set_botip(0,1,1);
582       set_botip(0,2,1);
583       set_boserif(0,1,4);
584       set_bosize(0,92);
585       push_stroke((point (2.3+mincho) of get_stroke(0))
586         {(1-2*mincho)*direction (2.3+mincho) of get_stroke(0)}..
587         (240,100)..(270,45+15*mincho)..(400,-10)..tension 3..(950,-20),
588         (1,1)-(1.1,1.1)-(1.1,1.1)-(1.7,1.7)-(1.9,1.9));
589       set_bosize(0,92);
590     endgroup;
591     expand_pbox;
592   endif;
593
594   vardef build_kanji.sscale(text tran)(text curves) =
595     tsu_xform(identity shifted (-centre_pt) tran shifted centre_pt)(curves);
596   endif;
597
598   vardef build_kanji.spread_legs(expr dist)(text curves) =

```

```

599 begingroup
600   save osp;
601   numeric osp;
602   osp:=sp;
603   curves;
604   save mytr;
605   transform mytr[];
606   (50,-50) transformed mytr1=(50,-50);
607   (500,-50) transformed mytr1=(500-dist/2,-50);
608   (500,850) transformed mytr1=(500-dist/2,850);
609   (950,-50) transformed mytr2=(950,-50);
610   (500,-50) transformed mytr2=(500+dist/2,-50);
611   (500,850) transformed mytr2=(500+dist/2,850);
612   save i;
613   i:=0;
614   forever:
615     exitif find_stroke(i)<osp;
616     if xpart 0.75[llcorner get_stroke(i),urcorner get_stroke(i)]<475:
617       replace_stroke(i,oldp transformed mytr1);
618     elseif xpart 0.25[llcorner get_stroke(i),urcorner get_stroke(i)]>525:
619       replace_stroke(i,oldp transformed mytr2);
620     fi;
621     i:=i-1;
622   endfor;
623 endgroup;
624 enddef;
625
626 vardef build_kanji.steam_enclose(expr ur)(text contents) =
627   push_pbox_toexpand("build_kanji.steam_enclose");
628   begingroup
629     save xfa,xfb,xfc,xfd;
630     transform xfa,xfb,xfc,xfd;
631     (50,-50) transformed xfc=(50,-50);
632     (950,850) transformed xfc=ur;
633     xypart xfc=yxpart xfc=0;
634     tsu_xform(xfc)(contents);
635
636     (0,0) transformed xfa=(0,950) transformed xfc;
637     (1,1) transformed xfa=(280,810);
638     xypart xfa=yxpart xfa=0;
639     (0,0) transformed xfb=(1100,950) transformed xfc;
640     (1,1) transformed xfb=(970,810);
641     xypart xfb=yxpart xfb=0;
642     (0,1) transformed xfd=(1100,950) transformed xfc;
643     (1,0) transformed xfd=(1000,-50);
644     xypart xfd=yxpart xfd=0;
645
646     push_stroke(((1,1)..tension 1.2..(0.5,0.45)..(0,0.2))) transformed xfa,

```

```

647     (1.7,1.7)–(1.5,1.5)–(1.2,1.2));
648 set_boserif(0,0,10);
649 set_bosize(0,90);
650
651 push_stroke((get_strokep(0) intersectionpoint
652     (((0,0.8)–(1,0.8)) transformed xfa))–
653     ((0.5,0.8) transformed xfb),
654     (1.5,1.5)–(1.6,1.6));
655 set_boserif(0,1,9);
656 set_bosize(0,90);
657
658 push_stroke(((0.8,0.4) transformed xfa)–((0.15,0.4) transformed xfb),
659     (1.6,1.6)–(1.6,1.6));
660 set_boserif(0,1,9);
661 set_bosize(0,90);
662
663 push_stroke(((0.2,0) transformed xfa)–(interpath(mincho,
664     (0,1)..tension 1.6..(0.4,0)..(0.6,0)..tension 1.5..
665     (0.73,0.2)..(0.8,0.4),
666     (0,1)..tension 1.6..(0.25,0.2)..{right}(0.8,0){curl 1}..
667     (0.6,0.2)..(0.6,0.4)
668     ) transformed xfd),
669     (1.6,1.6)–(1.6,1.6)–(1.4,1.4)–
670     (1.4,1.4)–(1.2,1.2)–(0.9,0.9));
671 set_boserif(0,1,4);
672 set_botip(0,1,1);
673 set_bosize(0,90);
674 endgroup;
675 expand_pbox;
676 enddef;
677
678 vardef build_kanji.tb(expr splitpoint,overlap)
679 (text topstuff)(text bottomstuff) =
680 perl_structure:=perl_structure
681     &"['build_kanji.tb',eids.u2ff1._1.1_',[";
682 begingroup
683     save t;
684     transform t[];
685     xypart t1=xypart t2=1;
686     xpart t1=xypart t1=yxpart t1=xpart t2=xypart t2=yxpart t2=0;
687     (0,850) transformed t1=(0,850);
688     (0,-50) transformed t1=(0,splitpoint-overlap/2);
689     (0,850) transformed t2=(0,splitpoint+overlap/2);
690     (0,-50) transformed t2=(0,-50);
691     tsu_xform(t1)(topstuff);
692     perl_structure:=perl_structure&"],[";
693     tsu_xform(t2)(bottomstuff);
694 endgroup;

```

**BUIL**

```

695 perl_structure:=perl_structure&"[]";
696 enddef;
697
698 vardef build_kanji.thickness(expr amount)(text curves) =
699   begingroup
700     save xsp;
701     xsp:=sp;
702     curves;
703     i:=0;
704     forever:
705       exitif find_stroke(i)<xsp;
706       if unknown get_bosize(i):
707         set_bosize(i,100);
708       fi;
709       if get_bosize(i)>0:
710         set_bosize(i,get_bosize(i)*amount);
711       fi;
712       i:=i+1;
713     endfor;
714   endgroup;
715 enddef;
716
717 vardef build_kanji.triclusterc(expr topxscale)
718   (text topstuff)(text leftstuff)(text rightstuff) =
719   build_kanji.tb(500,0)
720     (build_kanji.sscale(xscaled topxscale)(topstuff))
721     (build_kanji.lr(480,0)
722       (leftstuff)
723       (rightstuff));
724 enddef;
725
726 vardef build_kanji.wind_enclose(text contents) =
727   push_pbox_toexpand("build_kanji.wind_enclose");
728   push_stroke((50,50)..(120,100)..(160,300)..tension 1.2..(180,760)
729     -interpath(mincho,
730       (770,760){down}..{dir -72}(810,-20)..(860,-10)..tension 1.5..
731       (890,100)..(910,200),
732       (760,760){down}..(780,100)..{right}(910,-40){curl 1}..
733       (890,60)..(890,160)),
734     (1,1)-(1.3,1.3)-(1.5,1.5)-(1.6,1.6)-(1.6,1.6)-
735     (1.4,1.4)-(1.4,1.4)-(1.2,1.2)-(0.9,0.9));
736   set_botip(0,3,1);
737   set_botip(0,4,1);
738   set_boserif(0,3,4);
739   set_boserif(0,4,4);
740   begingroup
741     save t;
742     transform t;

```

```
743     (50,-50) transformed t=(230,-50);
744     (950,-50) transformed t=(700,-50);
745     (950,850) transformed t=(700,660+20*mincho);
746     tsu_xform(t)(contents);
747 endgroup;
748 expand_pbox;
749 enddef;
```

**BUIL**

## dakuten.mp

```
1 %
2 % Dakuten and handakuten for Tsukurimashou
3 % Copyright (C) 2011 Matthew Skala
4 %
5-29 [Standard copyright notice]
30
31 inclusion_lock(dakuten);
32
33 _____
34
35 vardef dakuten(expr xf) =
36   push_pbox_explicit("dakuten",
37     identity shifted (-0.4,-0.5) scaled 200 rotated -50 transformed xf);
38
39   push_stroke(((0,-80)..(-35,-35)..(10,-90)) transformed xf,
40     (1,1)..(1.4,1.4)..(1.8,1.8));
41   set_bosize(0,85);
42   set_boserif(0,2,4);
43
44   push_stroke(((0,80)..(50,30)..(100,-30)) transformed xf,
45     (1,1)..(1.4,1.4)..(1.8,1.8));
46   set_bosize(0,85);
47   set_boserif(0,2,4);
48 enddef;
49
50 vardef handakuten(expr location) =
51   push_lcblob(fullcircle scaled handakuten_outer shifted location
52     transformed inverse tsu_rescale_xform);
53   push_lcblob(reverse fullcircle scaled handakuten_inner shifted location
54     transformed inverse tsu_rescale_xform);
55 enddef;
```

DAKU

## enclosed.mp

```
1 %
2 % Enclosed characters for Tsukurimashou
3 % Copyright (C) 2011, 2012 Matthew Skala
4 %
5-29 [Standard copyright notice]
30
31 inclusion_lock(enclosed);
32
33 

---


34
35 vardef circle.single =
36   Fill fullcircle scaled (810+53*tsu_brush_max) shifted centre_pt;
37   unFill reverse fullcircle scaled (810-53*tsu_brush_max) shifted centre_pt;
38 enddef;
39
40 vardef circle.double =
41   Fill fullcircle scaled (880+60*tsu_brush_max) shifted centre_pt;
42   unFill reverse fullcircle scaled (880-20*tsu_brush_max) shifted centre_pt;
43   Fill fullcircle scaled (740+40*tsu_brush_max) shifted centre_pt;
44   unFill reverse fullcircle scaled (740-40*tsu_brush_max) shifted centre_pt;
45 enddef;
46
47 vardef square.single(text curves) =
48   tsu_xform(tsu_xf.sletter shifted tsu_xf.circ_slant_shift)(curves);
49   push_stroke(
50     ((500,790)-(100,790)-(100,10)-(900,10)-(900,790)-cycle)
51     transformed inverse tsu_slant_xform,
52     (2,2)-(2,2)-(2,2)-(2,2)-(2,2)-cycle);
53   set_bosize(0,80);
54   set_botip(0,1,1);
55   set_botip(0,2,1);
56   set_botip(0,3,1);
57   set_botip(0,4,1);
58 enddef;
59
60 

---


61
62 transform tsu_xf.circled;
63 xypart tsu_xf.circled=ypart tsu_xf.circled=0.68;
64 xypart tsu_xf.circled=yxpart tsu_xf.circled=0;
65 centre_pt transformed tsu_xf.circled=centre_pt;
66
67 transform tsu_xf.cletter;
68 xypart tsu_xf.cletter=ypart tsu_xf.cletter=0.56;
69 xypart tsu_xf.cletter=yxpart tsu_xf.cletter=0;
70 centre_pt transformed tsu_xf.cletter=centre_pt;
```

ENCL

```

71
72 transform tsu_xf.ctwo.left;
73 xypart tsu_xf.ctwo.left=yypart tsu_xf.ctwo.left=0.48;
74 xypart tsu_xf.ctwo.left=yxpart tsu_xf.ctwo.left=0;
75 (centre_pt+290*right) transformed tsu_xf.ctwo.left=centre_pt;
76
77 transform tsu_xf.ctwo.right;
78 xypart tsu_xf.ctwo.right=yypart tsu_xf.ctwo.right=0.48;
79 xypart tsu_xf.ctwo.right=yxpart tsu_xf.ctwo.right=0;
80 (centre_pt+310*left) transformed tsu_xf.ctwo.right=centre_pt;
81
82 transform tsu_xf.sletter;
83 xypart tsu_xf.sletter=yypart tsu_xf.sletter=0.71;
84 xypart tsu_xf.sletter=yxpart tsu_xf.sletter=0;
85 centre_pt transformed tsu_xf.sletter=centre_pt+10*up;
86
87 vardef tsu_xf.circ_slant_shift =
88   (centre_pt-(centre_pt transformed tsu_slant_xform))
89 enddef;
90
91 

---


92
93 transform tsu_xf.cbound;
94 tsu_xf.cbound=identity scaled 340 shifted centre_pt;

```

**ENCL**

## genjimon.mp

```
1 %
2 % Genjimon glyphs
3 % Copyright (C) 2011, 2012 Matthew Skala
4 %
5-29 [Standard copyright notice]
30
31 

---


32
33 genji_grid:=150;
34
35 if unknown tsu_brush_max:
36   if known brush_max:
37     tsu_brush_max:=brush_max;
38   else:
39     tsu_brush_max:=0.75;
40   fi;
41 fi;
42 if unknown genji_hw:
43   genji_hw:=tsu_brush_max/1.5;
44   if genji_hw>0.85: genji_hw:=0.85; fi;
45 fi;
46 if unknown genji_outline:
47   boolean genji_outline;
48   genji_outline:=false;
49 fi;
50 if genji_outline: genji_hw:=1-genji_hw; fi;
51 if unknown genji_rounded:
52   boolean genji_rounded;
53   genji_rounded:=false;
54 fi;
55
56 path genji_background;
57
58 % gb(f) - start a line at the bottom in file f
59 vardef gb(expr f,gp) =
60   begingroup
61     save myxf,mygl;
62     transform myxf;
63     path mygl;
64     myxf=identity scaled (genji_grid/2) shifted (whatever,whatever);
65     ((3-f)*2,4) transformed myxf=centre_pt;
66     if genji_rounded:
67       mygl:=((0,-genji_hw){right}..(genji_hw,0){up}..
68         {up}(gp shifted (0,1)){down}..
69         {down}(-genji_hw,0)..{right}cycle) transformed myxf;
70     else:
```

```

71     mygl:=((0-genji_hw)-(genji_hw-genji_hw)-
72         (gp shifted (0,1))-
73         (-genji_hw-genji_hw)-cycle) transformed myxf;
74 fi;
75 if genji_outline:
76     unFill reverse mygl;
77     save mybk,x,y,old_dir;
78     path mybk;
79     pair old_dir;
80     mybk:=(0,genji_hw-1.99) transformed myxf;
81     old_dir:=right;
82     for i:=1 upto (length mygl)-1:
83         numeric x[],y[];
84         z1=(point i of mygl)-(precontrol i of mygl);
85         z2=(postcontrol i of mygl)-(point i of mygl);
86         z3=z1/abs(z1);
87         z4=z2/abs(z2);
88         if z3=z4:
89             mybk:=mybk{old_dir}..
90                 {z3}((0.99-genji_hw)*genji_grid*(z4 rotated -90)
91                     +point i of mygl);
92         else:
93             mybk:=mybk{old_dir}..
94                 {z3}((0.99-genji_hw)*genji_grid*((z3+z4) rotated -90)
95                     +point i of mygl);
96         fi;
97         if (((point i of mybk)-(point (i-1) of mybk)) dotprod z3<0)
98         or (((point i of mybk)-(point (i-1) of mybk)) dotprod old_dir<0):
99             mybk:=(subpath (0,i-1) of mybk)-(point i of mybk);
100        fi;
101        if (length mybk)>3:
102            z5=(subpath ((length mybk)-4,(length mybk)-3) of mybk)
103                intersectiontimes
104                (subpath ((length mybk)-1,(length mybk)) of mybk);
105            if x5>0:
106                mybk:=(subpath (0,(length mybk)-4+x5) of mybk)..
107                    (subpath ((length mybk)-1+y5,infinity) of mybk)
108            fi;
109        fi;
110        if (length mybk)>3:
111            z6=(subpath ((length mybk)-4,(length mybk)-3) of mybk)
112                intersectiontimes
113                (subpath ((length mybk)-2,(length mybk)-1) of mybk);
114            if x6>0:
115                mybk:=(subpath (0,(length mybk)-4+x6) of mybk)..
116                    (subpath ((length mybk)-2+y6,infinity) of mybk)
117            fi;
118        fi;

```

```

119     if (length mybk)>3:
120         z7=((point (length mybk)-4 of mybk)
121             -(precontrol (length mybk)-4 of mybk));
122         z8=((postcontrol (length mybk)-3 of mybk)
123             -(point (length mybk)-3 of mybk));
124         if (abs(z7)>0) and (abs(z8)>0):
125             if (z7/abs(z7)) dotprod (z8/abs(z8))<=-0.1:
126                 mybk:=(subpath (0,(length mybk)-4) of mybk)-
127                     (subpath ((length mybk)-3,infinity) of mybk);
128             fi;
129         fi;
130     fi;
131     old_dir:=z4;
132 endfor;
133 mybk:=regenerate(mybk{old_dir}..{right}cycle);
134 dangerousFill mybk;
135 else:
136     Fill mygl;
137 fi;
138 endgroup;
139 enddef;
140
141 path ge_path[];
142 ge_path[0]=(genji_hw,0)-(genji_hw,genji_hw)-
143     (-genji_hw,genji_hw)-(-genji_hw,0);
144 ge_path[1]=(genji_hw,0){up}..(0,genji_hw){left}..
145     (-genji_hw,genji_hw)-(-genji_hw,0);
146 ge_path[2]=(genji_hw,0)-(genji_hw,genji_hw)-
147     (0,genji_hw){left}..{down}(-genji_hw,0);
148 ge_path[3]=(genji_hw,0){up}..(0,genji_hw){left}..{down}(-genji_hw,0);
149
150 % ge(t) - end a line, style t
151 vardef ge(expr t) =
152     if genji_rounded:
153         ((genji_hw,0)..(ge_path[t] shifted (0,1))..(-genji_hw,0))
154     else:
155         ((genji_hw,0)..(ge_path[0] shifted (0,1))..(-genji_hw,0))
156     fi
157 enddef;
158
159 % gf(d) - go forward d steps
160 vardef gf(expr d,gp) =
161     ((genji_hw,0)-(gp shifted (0,2*d))-(-genji_hw,0))
162 enddef;
163
164 % gr(r) - turn to right, radius r
165 vardef gr(expr r,gp) =
166     if genji_rounded and (r>=0):

```

```

167 ((genji_hw,0){up}..
168 (gp shifted (0,r+1) rotated -90 shifted (0,r+1))..
169 {down}{-genji_hw,0})
170 else:
171 ((genji_hw,0)-(genji_hw,max(r,0)+1-genji_hw)-
172 (gp shifted (0,max(r,0)+1) rotated -90 shifted (0,max(r,0)+1))-
173 (-genji_hw,max(r,0)+1+genji_hw)-(-genji_hw,0))
174 fi
175 enddef;
176
177 % gl(r) - turn to left, radius r
178 vardef gl(expr r,gp) =
179 if genji_rounded and (r>=0):
180 ((genji_hw,0)..
181 (gp shifted (0,r+1) rotated 90 shifted (0,r+1))..
182 (-genji_hw,0))
183 else:
184 ((genji_hw,0)-(genji_hw,max(r,0)+1+genji_hw)-
185 (gp shifted (0,max(r,0)+1) rotated 90 shifted (0,max(r,0)+1))-
186 (-genji_hw,max(r,0)+1-genji_hw)-(-genji_hw,0))
187 fi
188 enddef;
189
190 % gt(gpa,gpb) - make a T-junction
191 vardef gt(expr gpa,gpb) =
192 ((genji_hw,0)-(genji_hw,1-genji_hw)-
193 (gpa shifted (0,1) rotated -90 shifted (0,1))-
194 (genji_hw,1+genji_hw)-(gpb shifted (0,2))-(-genji_hw,0))
195 enddef;
196
197 % gx(gpa,gpb,gpc) - make an X-junction
198 vardef gx(expr gpa,gpb,gpc) =
199 ((genji_hw,0)-(genji_hw,1-genji_hw)-
200 (gpa shifted (0,1) rotated -90 shifted (0,1))-
201 (genji_hw,1+genji_hw)-(gpb shifted (0,2))-(-genji_hw,1+genji_hw)-
202 (gpc shifted (0,1) rotated 90 shifted (0,1))-
203 (-genji_hw,1-genji_hw)-(-genji_hw,0))
204 enddef;
205
206
207
208 % #1 Kiritsubo
209 vardef genjimon.kiritsubo =
210 gb(1,gf(2.5,gr(1,gr(1,gf(2.5,ge(3))))));
211 gb(2,gf(2,ge(3)));
212 gb(4,gf(3,gr(0,gr(0,gf(3,ge(3))))));
213 enddef;
214

```

```

215 % #2 Hahakigi
216 vardef genjimon.hahakigi =
217   gb(1,gf(3,ge(3)));
218   gb(2,gf(3,ge(3)));
219   gb(3,gf(3,ge(3)));
220   gb(4,gf(3,ge(3)));
221   gb(5,gf(3,ge(3)));
222 enddef;
223
224 % #3 Utsusemi
225 vardef genjimon.utsusemi =
226   gb(1,gf(3,ge(3)));
227   gb(2,gf(3,ge(3)));
228   gb(3,gf(3,ge(3)));
229   gb(4,gf(3,gr(0,gr(0,gf(3,ge(3))))));
230 enddef;
231
232 % #4 Yuugao
233 vardef genjimon.yuugao =
234   gb(1,gf(3,ge(3)));
235   gb(2,gf(3,ge(3)));
236   gb(3,gf(3,gr(0,gr(0,gf(3,ge(3))))));
237   gb(5,gf(3,ge(3)));
238 enddef;
239
240 % #5 Wakamurasaki
241 vardef genjimon.wakamurasaki =
242   gb(1,gf(3,ge(3)));
243   gb(2,gf(3,gr(0,gr(0,gf(3,ge(3))))));
244   gb(4,gf(3,gr(0,gr(0,gf(3,ge(3))))));
245 enddef;
246
247 % #6 Suetsumuhana
248 vardef genjimon.suetsumuhana =
249   gb(1,gf(3,gr(0,gt(gf(3,ge(3)),gt(gf(3,ge(3))),
250     gr(0,gf(3,ge(3))))));
251   gb(5,gf(3,ge(3)));
252 enddef;
253
254 % #7 Momiji no Ga
255 vardef genjimon.momiji_no_ga =
256   gb(1,gf(3,ge(3)));
257   gb(4,gf(2,ge(1)));
258   gb(2,gf(3,gr(0,gt(gf(3,ge(3)),gf(0.5,gr(1,gf(2.5,ge(3)))))));
259 enddef;
260
261 % #8 Hana no En
262 vardef genjimon.hana_no_en =

```

```

263 gb(1,gf(3,ge(3)));
264 gb(2,gf(3,ge(3)));
265 gb(3,gf(2.5,gr(1,gr(1,gf(2.5,ge(3))))));
266 gb(4,gf(2,ge(3)));
267 enddef;
268
269 % #9 Aoi
270 vardef genjimon.aoi =
271 gb(1,gf(3,gr(0,gr(0,gf(3,ge(3))))));
272 gb(3,gf(3,ge(3)));
273 gb(4,gf(3,ge(3)));
274 gb(5,gf(3,ge(3)));
275 enddef;
276
277 % #10 Sakaki
278 vardef genjimon.sakaki =
279 gb(1,gf(3,gr(0,gt(gf(3,ge(3)),gr(0,gf(3,ge(3))))));
280 gb(4,gf(3,gr(0,gr(0,gf(3,ge(3))))));
281 enddef;
282
283 % #11 Hana Chiru Sato
284 vardef genjimon.hana_chiru_sato =
285 gb(1,gf(3,ge(3)));
286 gb(2,gf(2,gr(2,gr(0,gx(gl(0,gf(2,ge(3))),
287 gf(2,ge(3)),gr(0,gf(2,ge(3))))));
288 enddef;
289
290 % #12 Suma
291 vardef genjimon.suma =
292 gb(1,gf(2,gr(0,gx(gf(2,ge(3)),gt(gf(2,ge(3)),
293 gr(0,gf(2,ge(3))),gr(0,gf(1,gr(2,gf(2,ge(3))))));
294 enddef;
295
296 % #13 Akashi
297 vardef genjimon.akashi =
298 gb(1,gf(3,ge(3)));
299 gb(2,gf(3,gr(0,gr(0,gf(3,ge(3))))));
300 gb(4,gf(3,ge(3)));
301 gb(5,gf(3,ge(3)));
302 enddef;
303
304 % #14 Miotsukushi
305 vardef genjimon.miotsukushi =
306 gb(1,gf(3,ge(3)));
307 gb(2,gf(2.5,gr(1,gf(0.5,gt(gf(3,ge(3)),gr(0,gf(3,ge(3))))));
308 gb(3,gf(2,ge(2)));
309 enddef;
310

```

```

311 % #15 Yomogyuu
312 vardef genjimon.yomogyuu =
313   gb(1,gf(3,gr(0,gt(gf(3,ge(3)),gr(0,gf(3,ge(3)))))));
314   gb(4,gf(3,ge(3)));
315   gb(5,gf(3,ge(3)));
316 enddef;
317
318 % #16 Sekiya
319 vardef genjimon.sekiya =
320   gb(1,gf(3,ge(3)));
321   gb(2,gf(3,gr(0,gt(gf(3,ge(3)),gr(0,gf(3,ge(3)))))));
322   gb(5,gf(3,ge(3)));
323 enddef;
324
325 % #17 Eawase
326 vardef genjimon.eawase =
327   gb(1,gf(2,gr(0,gx(gf(2,ge(3)),gr(-1,gf(2,ge(3))),
328     gr(0,gf(1.5,gr(1,gf(2.5,ge(3)))))));
329   gb(4,gf(2,ge(1)));
330 enddef;
331
332 % #18 Matsukaze
333 vardef genjimon.matsukaze =
334   gb(1,gf(3,gr(0,gr(0,gf(3,ge(3))))));
335   gb(3,gf(3,gr(0,gr(0,gf(3,ge(3))))));
336   gb(5,gf(3,ge(3)));
337 enddef;
338
339 % #19 Usugumo
340 vardef genjimon.usugumo =
341   gb(1,gf(3,ge(3)));
342   gb(2,gf(3,gr(0,gt(gf(3,ge(3)),gt(gf(3,ge(3)),
343     gr(0,gf(3,ge(3)))))));
344 enddef;
345
346 % #20 Asagao
347 vardef genjimon.asagao =
348   gb(1,gf(2.5,gr(1,gf(0.5,gt(gf(3,ge(3)),gr(0,gf(3,ge(3)))))));
349   gb(2,gf(2,ge(2)));
350   gb(5,gf(3,ge(3)));
351 enddef;
352
353 % #21 Otome
354 vardef genjimon.otome =
355   gb(1,gf(2.5,gr(1,gr(1,gf(2.5,ge(3))))));
356   gb(2,gf(2,ge(3)));
357   gb(4,gf(3,ge(3)));
358   gb(5,gf(3,ge(3)));

```

```

359 enddef;
360
361 % #22 Tamakazura
362 vardef genjimon.tamakazura =
363   gb(1,gf(3,gr(0,gr(0,gf(3,ge(3))))));
364   gb(3,gf(3,gr(0,gt(gf(3,ge(3)),gr(0,gf(3,ge(3))))));
365 enddef;
366
367 % #23 Hatsune
368 vardef genjimon.hatsune =
369   gb(1,gf(2,gr(0,gx(gf(2,ge(3)),gr(0,gf(2,ge(3))),
370     gr(0,gr(2,gf(2,ge(3))))));
371   gb(5,gf(3,ge(3)));
372 enddef;
373
374 % #24 Kochou
375 vardef genjimon.kochou =
376   gb(1,gf(2,gr(2,gf(1,gr(0,gx(reverse gt(gf(2,ge(3)),
377     gr(0,gf(2,ge(3)))) xscaled -1,gf(2,ge(3)),
378     gr(0,gf(2,ge(3))))));
379 enddef;
380
381 % #25 Hotaru
382 vardef genjimon.hotaru =
383   gb(1,gf(3,gr(0,gt(gf(3,ge(3)),gf(0.5,gr(1,gf(2.5,ge(3))))));
384   gb(3,gf(2,ge(1)));
385   gb(5,gf(3,ge(3)));
386 enddef;
387
388 % #26 Tokonatsu
389 vardef genjimon.tokonatsu =
390   gb(1,gf(3,ge(3)));
391   gb(2,gf(3,ge(3)));
392   gb(3,gf(3,gr(0,gt(gf(3,ge(3)),gr(0,gf(3,ge(3))))));
393 enddef;
394
395 % #27 Kagaribi
396 vardef genjimon.kagaribi =
397   gb(1,gf(3,ge(3)));
398   gb(2,gf(2.5,gr(1,gr(1,gf(2.5,ge(3))))));
399   gb(3,gf(2,ge(3)));
400   gb(5,gf(3,ge(3)));
401 enddef;
402
403 % #28 Nowaki
404 vardef genjimon.nowaki =
405   gb(1,gf(3,gr(0,gr(0,gf(3,ge(3))))));
406   gb(3,gf(3,ge(3)));

```

```

407 gb(4,gf(3,gr(0,gr(0,gf(3,ge(3))))));
408 enddef;
409
410 % #29 Miyuki
411 vardef genjimon.miyuki =
412 gb(1,gf(2,gr(2,gr(0,gx(gl(0,gf(2,ge(3))),
413 gf(2,ge(3)),gt(gf(2,ge(3)),gr(0,gf(2,ge(3))))))));
414 enddef;
415
416 % #30 Fujibakama
417 vardef genjimon.fujibakama =
418 gb(1,gf(2.5,gr(1,gf(1,gr(1,gf(2.5,ge(3))))));
419 gb(2,gf(2,ge(2)));
420 gb(3,gf(2,ge(1)));
421 gb(5,gf(3,ge(3)));
422 enddef;
423
424 % #31 Makibashira
425 vardef genjimon.makibashira =
426 gb(1,gf(1.5,gr(3,gr(3,gf(1.5,ge(3))))));
427 gb(2,gf(1.5,gr(1,gr(1,gf(1.5,ge(3))))));
428 gb(3,gf(1,ge(3)));
429 enddef;
430
431 % #32 Umegae
432 vardef genjimon.umegae =
433 gb(1,gf(3,gr(0,gt(gf(3,ge(3)),gt(gf(3,ge(3)),gf(0.5,
434 gr(1,gf(2.5,ge(3))))));
435 gb(4,gf(2,ge(1)));
436 enddef;
437
438 % #33 Fuji no Uraba
439 vardef genjimon.fuji_no_uraba =
440 gb(1,gf(3,ge(3)));
441 gb(2,gf(2,gr(2,gr(2,gf(2,ge(3))))));
442 gb(3,gf(2,gr(0,gr(0,gf(2,ge(3))))));
443 enddef;
444
445 % #34 Wakana no Jou
446 vardef genjimon.wakana_no_jou =
447 gb(1,gf(3,gr(0,gt(gf(3,ge(3)),gf(1,
448 gr(2,gf(2,ge(3))))));
449 gb(3,gf(2,gr(-1,gr(0,gf(2,ge(3))))));
450 enddef;
451
452 % #35 Wakana no Ge
453 vardef genjimon.wakana_no_ge =
454 gb(1,gf(3,gr(0,gt(gf(3,ge(3)),gf(1,

```

```

455     gr(0,gx(gl(-1,gf(2,ge(3))),gf(2,ge(3)),gr(0,gf(2,ge(3))))))));
456 enddef;
457
458 % #36 Kashiwagi
459 vardef genjimon.kashiwagi =
460   gb(1,gf(2.5,gr(1,gf(0.5,gt(gf(3,ge(3)),
461     gf(0.5,gr(1,gf(2.5,ge(3))))))));
462   gb(2,gf(2,ge(2)));
463   gb(4,gf(2,ge(1)));
464 enddef;
465
466 % #37 Yokobue
467 vardef genjimon.yokobue =
468   gb(1,gf(2.5,gr(1,gf(1.5,gt(gf(3,ge(3)),
469     gr(0,gf(3,ge(3))))))));
470   gb(2,gf(2,ge(2)));
471   gb(3,gf(2,ge(0)));
472 enddef;
473
474 % #38 Suzumushi
475 vardef genjimon.suzumushi =
476   gb(1,gf(2.5,gr(1,gf(1.5,gr(2,gf(2,ge(3))))));
477   gb(2,gf(2,ge(2)));
478   gb(3,gf(2,gr(-1,gr(0,gf(2,ge(3))))));
479 enddef;
480
481 % #39 Yuugiri
482 vardef genjimon.yuugiri =
483   gb(1,gf(1.5,gr(1,gf(0.5,gx(gf(2,ge(3)),gr(0,gf(2,ge(3))),
484     gr(0,gr(2,gf(2,ge(3))))))));
485   gb(2,gf(1,ge(2)));
486 enddef;
487
488 % #40 Minori
489 vardef genjimon.minori =
490   gb(1,gf(1.5,gr(3,gf(0.5,gr(0,gx(gf(0.5,gl(1,gf(1.5,ge(3))),
491     gf(2,ge(3)),gr(0,gf(2,ge(3))))))));
492   gb(3,gf(1,ge(2)));
493 enddef;
494
495 % #41 Maboroshi
496 vardef genjimon.maboroshi =
497   gb(1,gf(2.5,gr(1,gf(2,gr(1,gf(2.5,ge(3))))));
498   gb(2,gf(2,ge(2)));
499   gb(3,gf(2,ge(0)));
500   gb(4,gf(2,ge(1)));
501 enddef;
502

```

```

503 % #42 Ninounomiya
504 vardef genjimon.ninounomiya =
505   gb(1,gf(2,gr(0,gt(gf(2,ge(3)),gx(gf(2,ge(3)),gr(0,gf(2,ge(3))),
506     gr(0,gr(2,gf(2,ge(3)))))))));
507 enddef;
508
509 % #43 Koubai
510 vardef genjimon.koubai =
511   gb(1,gf(3,ge(3)));
512   gb(2,gf(2.5,gr(1,gf(1,gr(1,gf(2.5,ge(3)))))));
513   gb(3,gf(2,ge(2)));
514   gb(4,gf(2,ge(1)));
515 enddef;
516
517 % #44 Takegawa
518 vardef genjimon.takegawa =
519   gb(1,gf(2,gr(2,gf(1,gr(2,gf(2,ge(3)))))));
520   gb(2,gf(2,gr(0,gt(gf(2,ge(3)),gr(0,gf(2,ge(3)))))));
521 enddef;
522
523 % #45 Hashihime
524 vardef genjimon.hashihime =
525   gb(1,gf(2.5,gr(1,gf(0.5,gt(gf(3,ge(3)),gt(gf(3,ge(3)),
526     gr(0,gf(3,ge(3)))))));
527   gb(2,gf(2,ge(2)));
528 enddef;
529
530 % #46 Shii ga Moto
531 vardef genjimon.shii_ga_moto =
532   gb(1,gf(2,gr(2,gr(2,gf(2,ge(3))))));
533   gb(2,gf(2,gr(0,gr(0,gf(2,ge(3))))));
534   gb(5,gf(3,ge(3)));
535 enddef;
536
537 % #47 Agemaki
538 vardef genjimon.agemaki =
539   gb(1,gf(2,gr(2,gf(1,gt(gf(3,ge(3)),gr(0,gf(3,ge(3)))))));
540   gb(2,gf(2,gr(0,gr(-1,gf(2,ge(3))))));
541 enddef;
542
543 % #48 Sawarabi
544 vardef genjimon.sawarabi =
545   gb(1,gf(3,gr(0,gr(0,gf(3,ge(3))))));
546   gb(3,gf(2.5,gr(1,gr(1,gf(2.5,ge(3))))));
547   gb(4,gf(2,ge(3)));
548 enddef;
549
550 % #49 Yadorigi

```

```

551 vardef genjimon.yadorigi =
552   gb(1,gf(3,gr(0,gt(gf(3,ge(3)),gf(1,gt(gf(3,ge(3)),
553     gr(0,gf(3,ge(3)))))))));
554   gb(3,gf(2,ge(0)));
555 enddef;
556
557 % #50 Azumaya
558 vardef genjimon.azumaya =
559   gb(1,gf(3,gr(0,gt(gf(3,ge(3)),gf(1.5,gr(1,gf(2.5,ge(3)))))))));
560   gb(3,gf(2,ge(0)));
561   gb(4,gf(2,ge(1)));
562 enddef;
563
564 % #51 Ukifune
565 vardef genjimon.ukifune =
566   gb(1,gf(2,gr(2,gf(1,gf(0.5,gr(1,gf(2.5,ge(3)))))))));
567   gb(2,gf(2,gr(0,gr(-1,gf(2,ge(3))))));
568   gb(4,gf(2,ge(1)));
569 enddef;
570
571 % #52 Kagerou
572 vardef genjimon.kagerou =
573   gb(1,gf(2,gr(2,gt(gx(gl(0,gf(2,ge(3))),
574     gf(2,ge(3)),gr(0,gf(2,ge(3))),gr(2,gf(2,ge(3)))))));
575 enddef;
576
577 % #53 Tenarai
578 vardef genjimon.tenarai =
579   gb(1,gf(3,gr(0,gt(gf(3,ge(3)),gt(gf(3,ge(3)),gt(gf(3,ge(3)),
580     gr(0,gf(3,ge(3)))))))));
581 enddef;
582
583 % #54 Yume no Ukihashi
584 vardef genjimon.yume_no_ukihashi =
585   gb(1,gf(3,gr(0,gr(0,gf(3,gl(0,gl(0,gf(3,gr(0,gr(0,gf(3,
586     gl(0,gl(0,gf(3,ge(3)))))))))))));
587 enddef;

```

# hiragana.mp

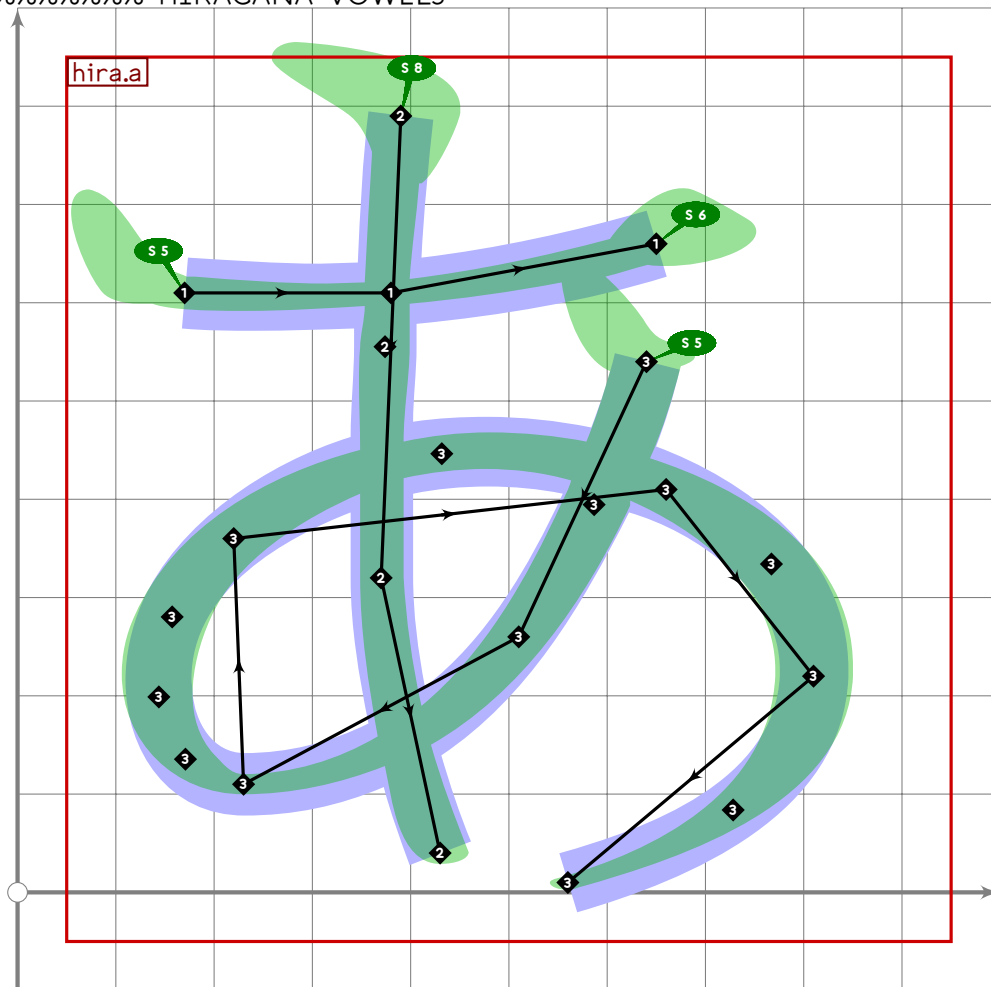
```

1 %
2 % Hiragana for Tsukurimashou
3 % Copyright (C) 2011, 2012 Matthew Skala
4 %
5-29 [Standard copyright notice]
30
31 inclusion_lock(hiragana);
32
33
34

```

## Hiragana Vowels

35 %%%%%%%%%% HIRAGANA VOWELS



HIRA

```

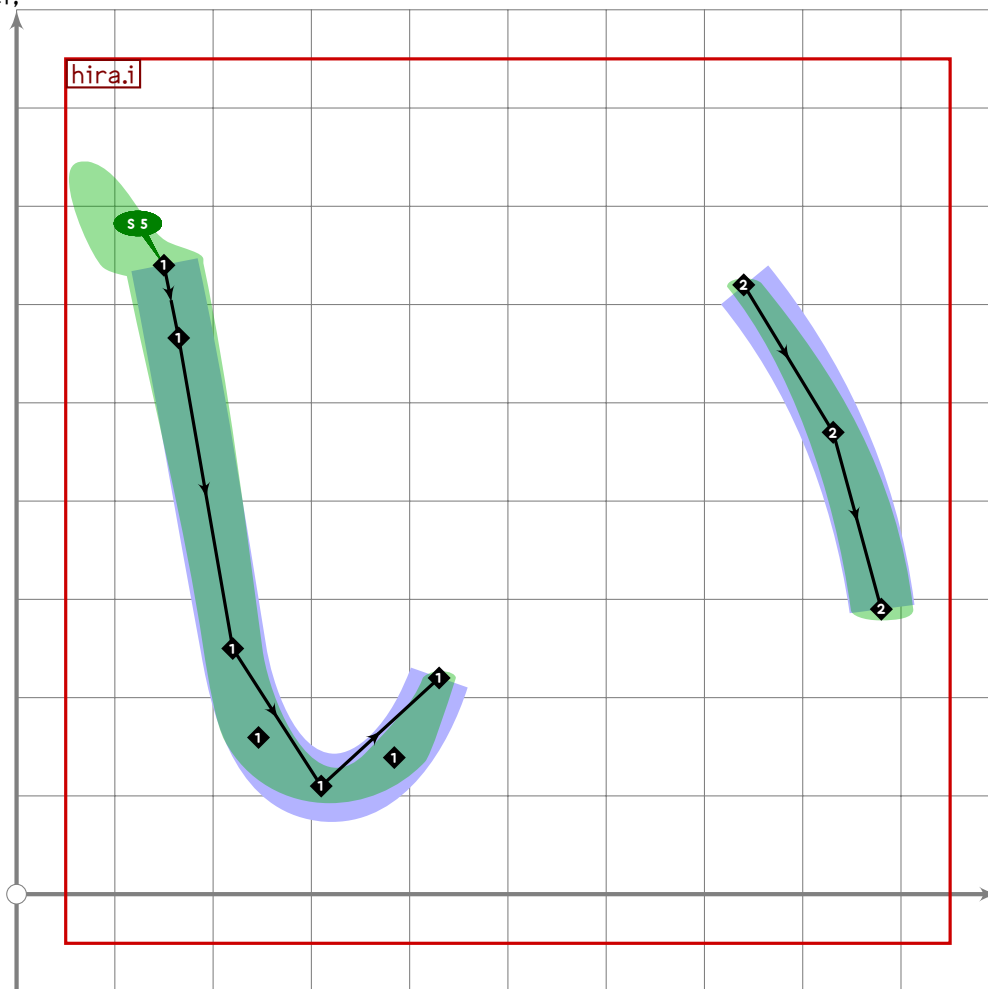
36
37 vardef hira.a =
38   push_pbox_toexpand("hira.a");
39
40   push_stroke((170,610)..(380,610)..(650,660),
41     (1.6,1.7)-(1.4,1.4)-(1.6,1.6));

```

```

42 set_boserif(0,0,5);
43 set_boserif(0,2,6);
44
45 push_stroke((390,790)..tension 1.5..(370,320)..(430,40),
46   (1.2,1.2)-(1.1,1.1)-(1.3,1.3));
47 set_boserif(0,0,8);
48
49 push_stroke((640,540)..tension 1.2..(510,260)..(230,110){left}..
50   (220,360)..(660,410)..(810,220)..{curl 0}(560,10),
51   (1.5,1.5)-(1.4,1.4)-(1.2,1.2)-
52   (1.7,1.7)-(1.8,1.8)-(1.6,1.6)-(1,1));
53 set_boserif(0,0,5);
54 expand_pbox;
55 enddef;

```



```

56
57 vardef hira.i =
58   push_pbox_toexpand("hira.i");
59
60   push_stroke((150,640)..(165,566)..(220,250)..(310,110)..{curl 0.1}(430,220),
61     (1.6,1.6)..(1.6,1.6)..(1.3,1.3)..(1.8,1.8)..(1,1));
62   set_boserif(0,0,5);

```

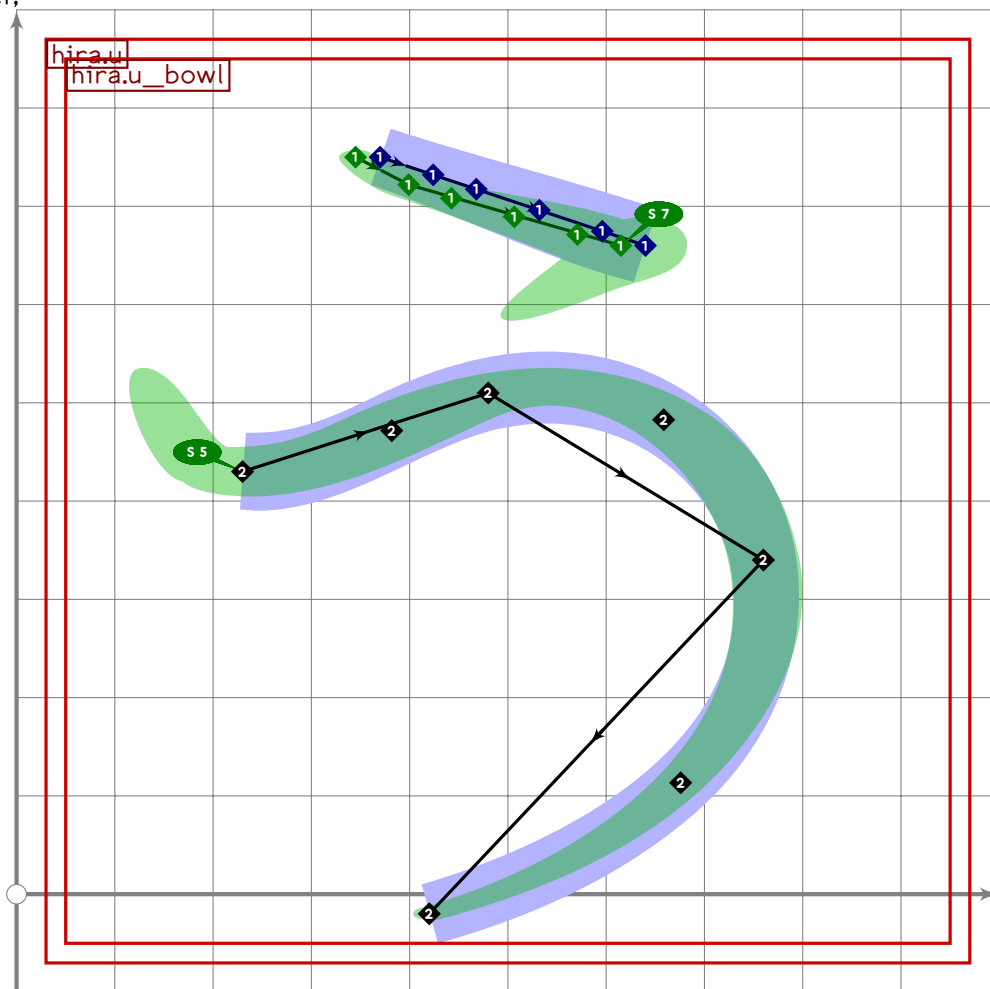
HIRA

U+3046  
tsuku.uni3046

```

63
64  push_stroke((740,620)..(831,470)..(880,290),
65    (1,1)..(1.3,1.3)..(1.4,1.4));
66  expand_pbox;
67 enddef;
68
69 vardef hira.u_bowl =
70  push_pbox_toexpand("hira.u_bowl");
71
72  push_stroke((230,430){dir 355}..(480,510)..(760,340)..{curl 0.1}(420,-20),
73    (2.3,2.3)..(2,2)..(1.5,1.5)..(1,1));
74  set_boserif(0,0,5);
75  expand_pbox;
76 enddef;

```



```

77
78 vardef hira.u =
79  push_pbox_toexpand("hira.u");
80
81  push_stroke(((370,750)..(0.2[(370,750),(640,660)]+10*down*mincho)..
82    tension 2..(640,660)) shifted (25*left*mincho),
83    (1,1)..(1.4,1.4)..(2.3,2.3));

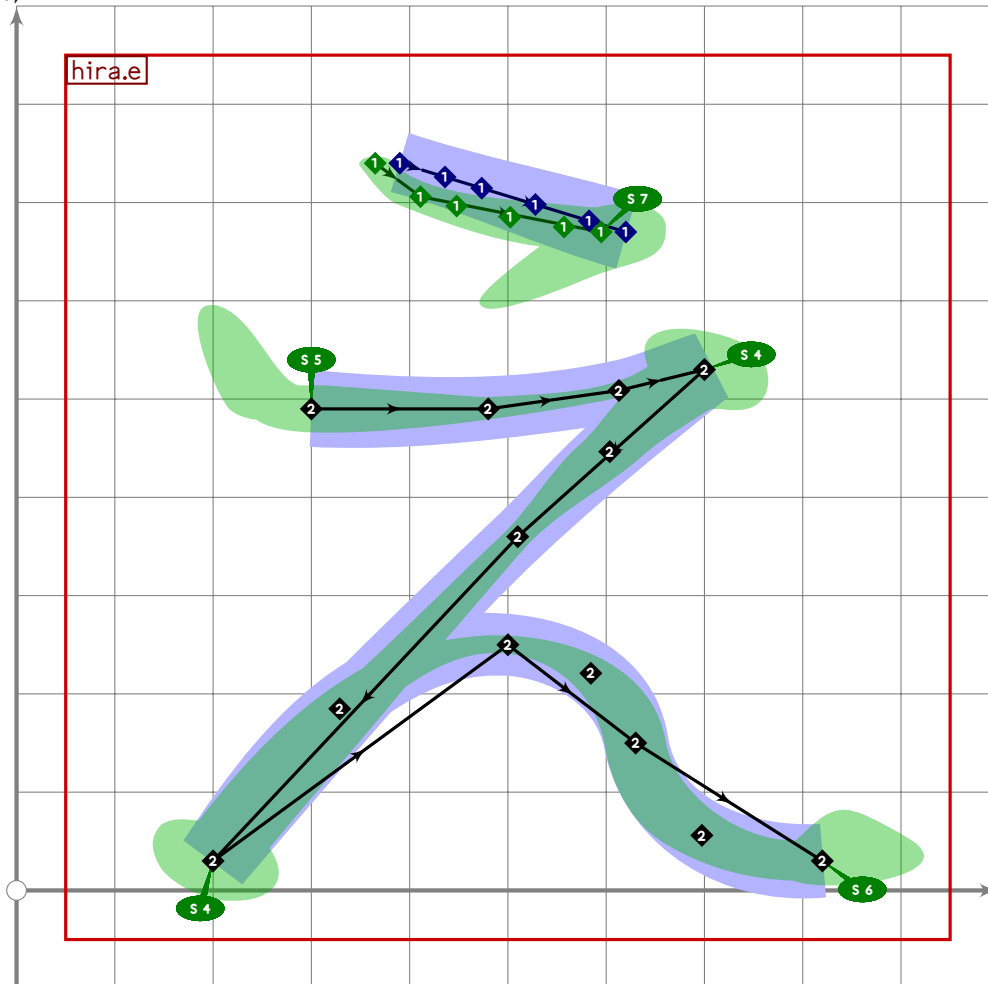
```

HIRA

```

84 set_boserif(0,2,7);
85
86 hira.u_bowl;
87 expand_pbox;
88 enddef;

```



```

89
90 vardef hira.e =
91   push_pbox_toexpand("hira.e");
92
93   push_stroke(((390,740)..(0.2[(390,740),(620,670)]+20*down*mincho)..
94     tension 2..(620,670)) shifted (25*left*mincho,
95     (1,1)..(14,14)..(2.3,2.3));
96   set_boserif(0,2,7);
97
98   push_stroke((300,490)..(480,490)..{curl 1}(700,530){curl 1}..
99     (510,360)..{curl 1}(200,30){curl 0}..
100     (500,250)..(630,150){dir 280}{dir 5}(820,30),
101     (2.2,2.2)-(1.3,1.3)-(1,1)-(2.01,2.01)
102     -(1,1)-(1.7,1.7)
103     -(1.2,1.2)-(14,14)-(2,2));
104   replace_strokep(0)(insert_nodes(olddp)(1.6));

```

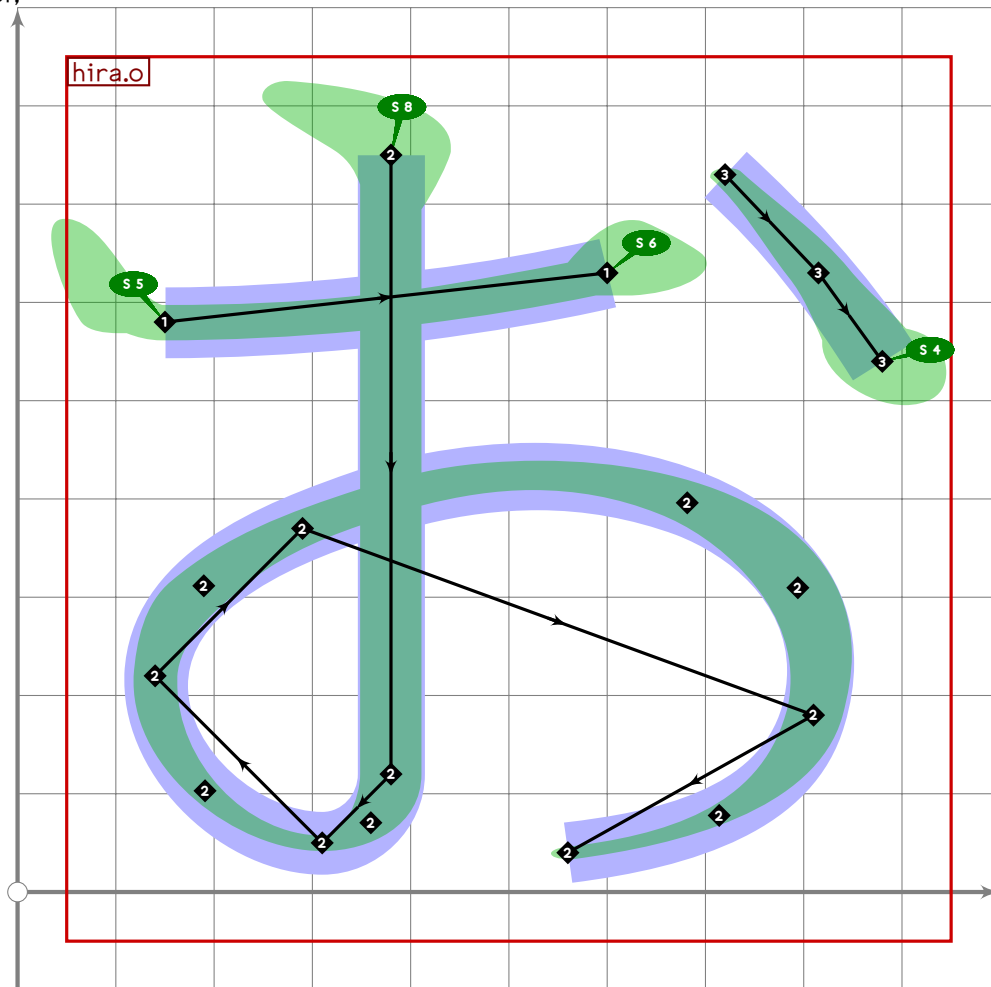
**HIRA**

U+304A  
tsuku.uni304A

```

105 set_botip(0,3,0);
106 set_botip(0,5,0);
107 set_boserif(0,0,5);
108 set_boserif(0,3,4);
109 set_boserif(0,5,4);
110 set_boserif(0,8,6);
111 expand_pbox;
112 enddef;

```



## HIRA

```

113
114 vardef hira.o =
115   push_pbox_toexpand("hira.o");
116
117   push_stroke((150,580){right}..(600,630),
118     (1.8,1.8)..(1.8,1.8));
119   set_boserif(0,0,5);
120   set_boserif(0,1,6);
121
122   push_stroke((380,750)..(380,120){down}..(310,50){left}..tension 1.1..
123     (140,220)..(290,370)..(810,180){curl 0}(560,40),
124     (1.4,1.4)..(1.3,1.3)..(1,1)..(1.5,1.5)..(1.6,1.6)..
125     (1.6,1.6)..(1,1));

```

```

126 set_boserif(0,0,8);
127
128 push_stroke((720,730)..(815,630)..(880,540),
129   (1,1)..(14,14)..(18,18));
130 set_boserif(0,2,4);
131 expand_pbox;
132 enddef;
133

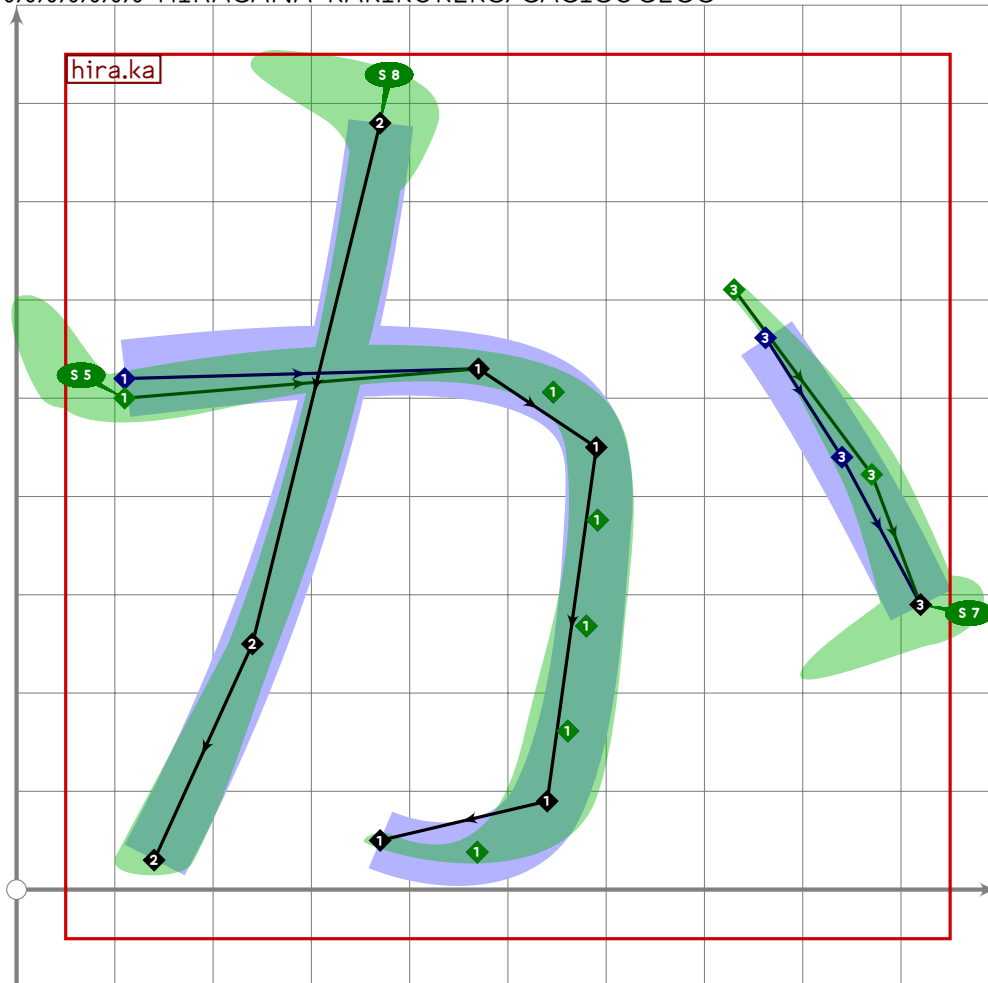
```

## Hiragana Kakikukeko/Gagigugego

```

134 %%%%%%%%% HIRAGANA KAKIKUKEKO/GAGIGUGEGO

```



HIRA

```

135
136 vardef hira.ka =
137   push_pbox_toexpand("hira.ka");
138
139   push_stroke(((110,520)+20*mincho*down){curl 0}..(470,530)..(590,450)..
140     tension 2..(540,90)..{curl 0.3}(370,50),
141     (2.3,2.3)..(1.7,1.7)..(14,14)..(18,18)..(1,1));
142   set_boserif(0,0,5);
143
144   push_stroke((370,780)..(240,250)..(140,30),

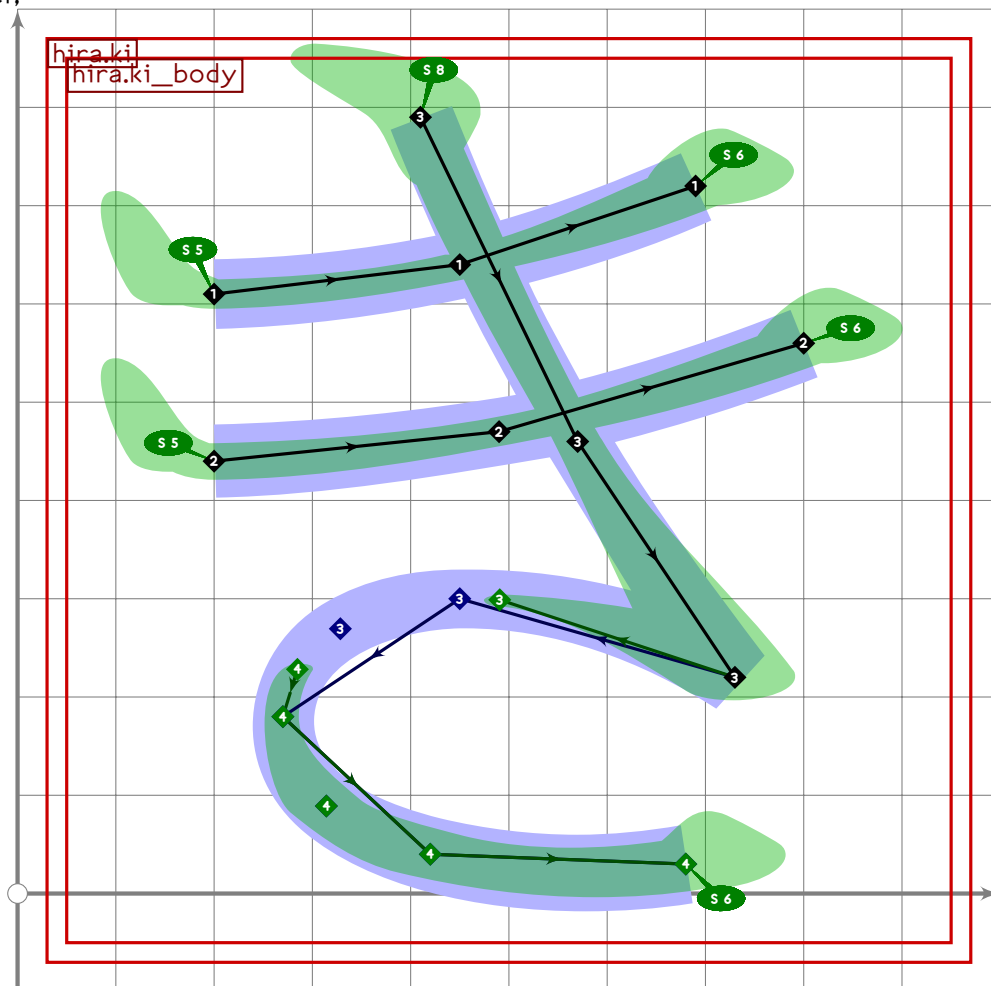
```

U+304D  
tsuku.uni304D

```

145   (1,3,1,3)..(1,2,1,2)..(1,6,1,6));
146   set_boserif(0,0,8);
147
148   push_stroke((720,620)..((840,440)+35*mincho*(dir -30))..(920,290),
149   (0,8,0,8)..(1,4,1,4)..(1,6,1,6));
150   set_boserif(0,2,7);
151   expand_pbox;
152 enddef;
153
154 vardef hira.ki_body =
155   push_pbox_toexpand("hira.ki_body");
156
157   push_stroke((410,790)..(570,460)..{curl 1}{730,220}{curl 1}..
158   (450,300)..(270,180)..(420,40)..(680,30),
159   (1,4,1,4)-(1,2,1,2)-(2,3,2)-(0,6,0,1)..(0,9,1,1)..
160   (2,1,2,1)..(2,4,2,4));
161   set_botip(0,2,0);
162   set_boserif(0,0,8);
163   set_boserif(0,6,6);
164   expand_pbox;
165 enddef;

```

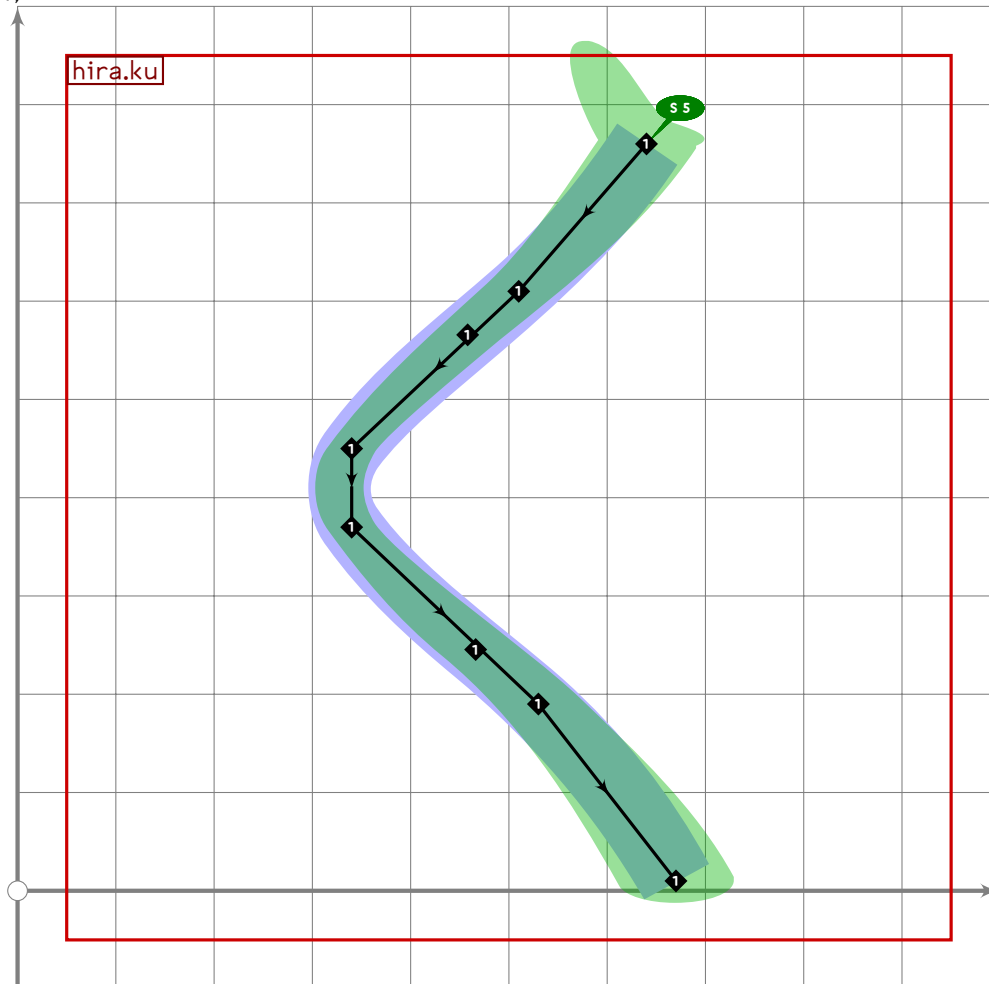


HIRA

```

166
167 vardef hira.ki =
168   push_pbox_toexpand("hira.ki");
169
170   push_stroke((200,610)..(450,640)..(690,720),
171     (1,6,1,6)–(1,4,1,4)–(1,9,1,9));
172   set_boserif(0,0,5);
173   set_boserif(0,2,6);
174
175   push_stroke((200,440)..(490,470)..(800,560),
176     (1,9,1,9)–(1,5,1,5)–(1,9,1,9));
177   set_boserif(0,0,5);
178   set_boserif(0,2,6);
179
180   hira.ki_body;
181   expand_pbox;
182 enddef;

```



```

183
184 vardef hira.ku =
185   push_pbox_toexpand("hira.ku");
186

```

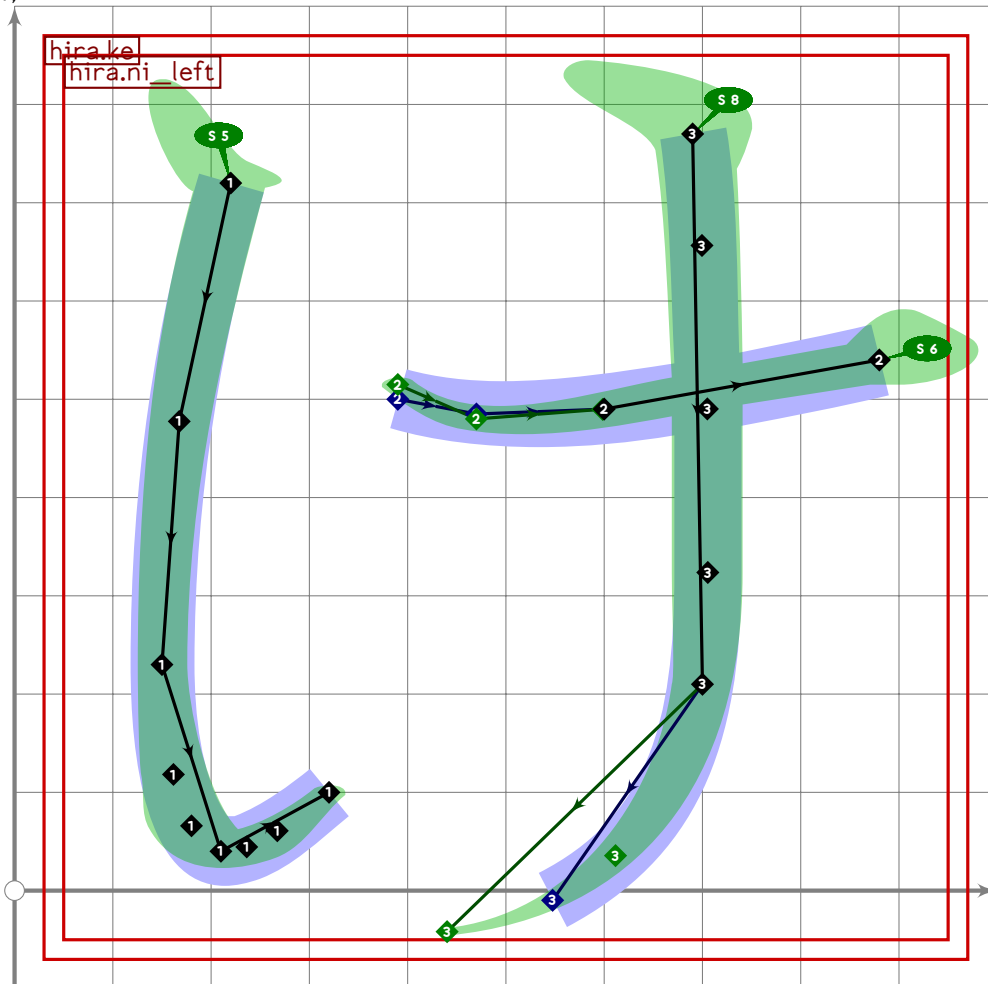
HIRA

U+3051  
tsuku.uni3051

```

187 push_stroke((640,760)..(510,610)..(340,450)..
188     tension 0.75..(340,370)..(530,190)..(670,10),
189     (1,9,19)..(1,6,1,6)..(1,2,1,2)..(1,2,1,2)..(1,7,1,7)..(2,1,2,1));
190 set_boserif(0,0,5);
191 expand_pbox;
192 enddef;

```

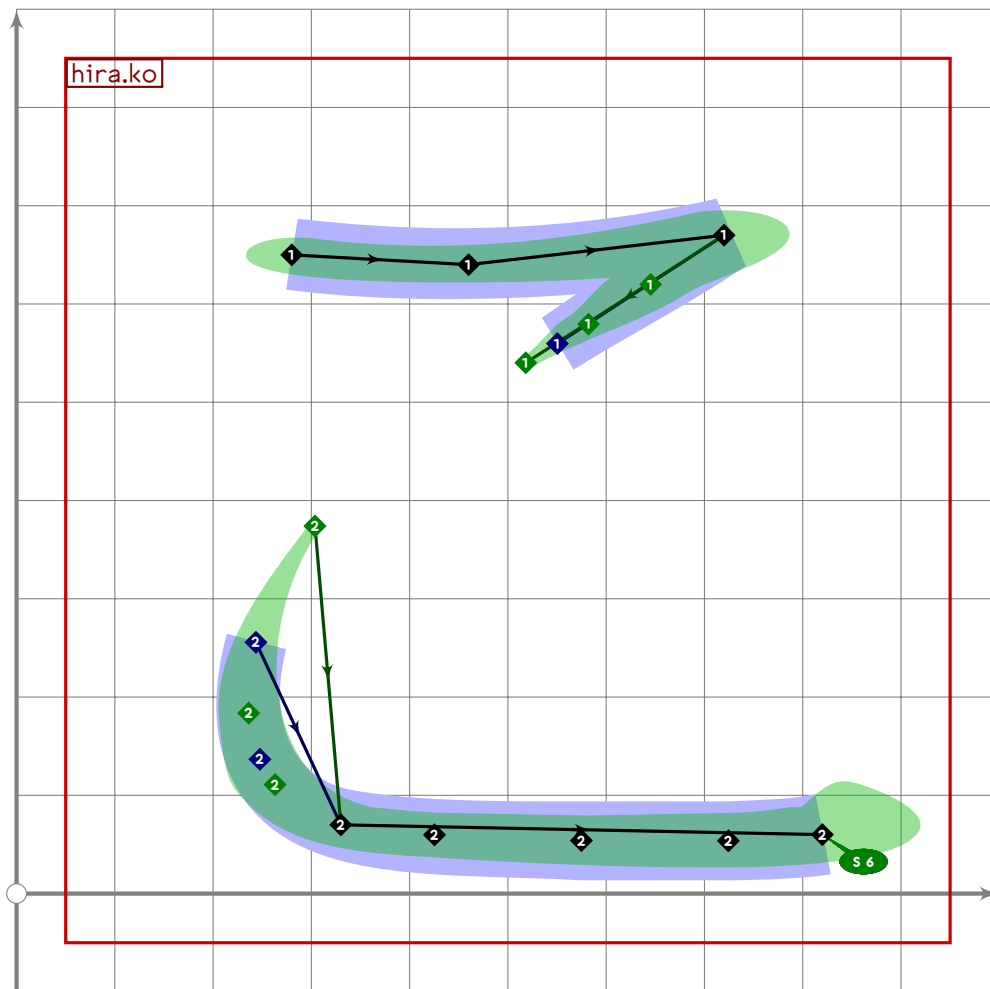


```

193
194 vardef hira.ke =
195   push_pbox_toexpand("hira.ke");
196
197   hira.ni_left;
198
199   push_stroke((390,500+15*mincho)..(470,485-5*mincho)..(600,490)..(880,540),
200     (1,1)..(1,4,1,4)..(1,8,1,8)..(2,2));
201   set_boserif(0,3,6);
202
203   push_stroke((690,770)..tension 2..(700,210)..(280,-10),
204     (1,6,1,6)-(1,4,1,4)-(0,6,0,6));
205   set_boserif(0,0,8);
206   expand_pbox;
207 enddef;

```

**HIRA**



```

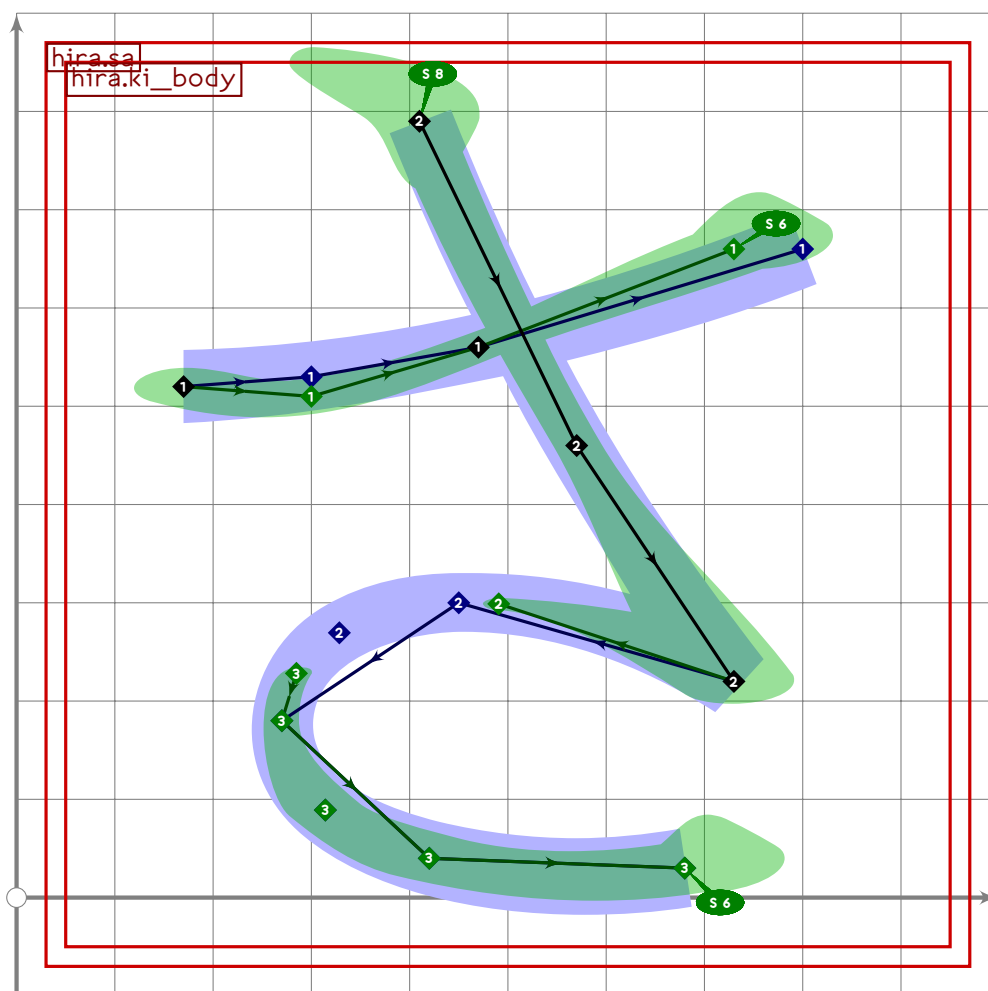
208
209 vardef hira.ko =
210   push_pbox_toexpand("hira.ko");
211
212   push_stroke((280,650)..(460,640)..{curl 1}(720,670){curl 1}..
213     (450,500)..(330,70)..tension 2.4..(820,60),
214     (1.8,1.8)-(1.9,1.9)-(2.3,2.3)-
215     (0.35,0.25)-(2.2,1.8)..(2.8,2.4));
216   set_botip(0,2,0);
217   set_boserif(0,5,6);
218   expand_pbox;
219 enddef;
220

```

**HIRA**

Hiragana Sashisuseso/Zajizuzezo

221 %%%%%%%%% HIRAGANA SASHISUSES0/ZAJIZUZEZO

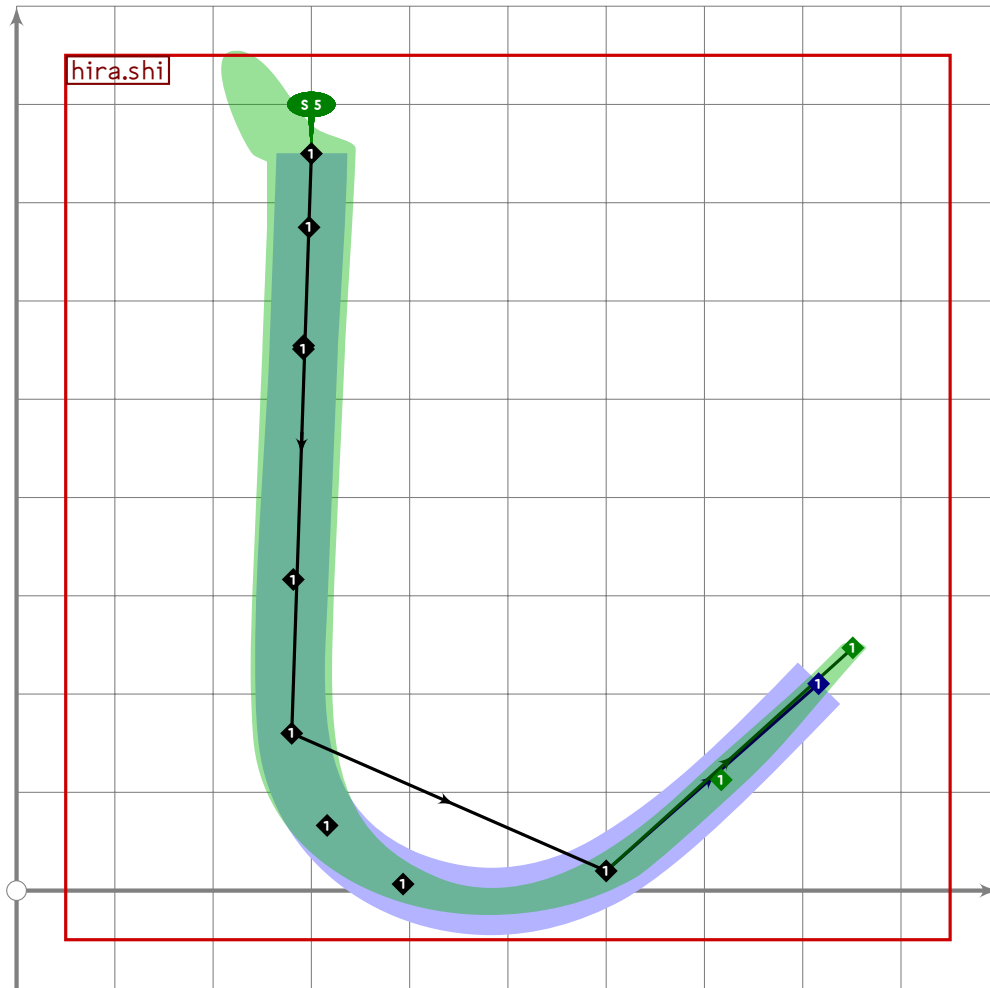


```

222
223 vardef hira.sa =
224   push_pbox__toexpand("hira.sa");
225
226   push_stroke((170,520)..(300,530-20*mincho)..(470,560)..(800-70*mincho,660),
227     (1,1,9)-(1,8,1,8)-(1,4,1,4)-(2,1,2,1));
228   set_boserif(0,3,6);
229
230   hira.ki_body;
231   expand_pbox;
232 enddef;

```

**HIRA**

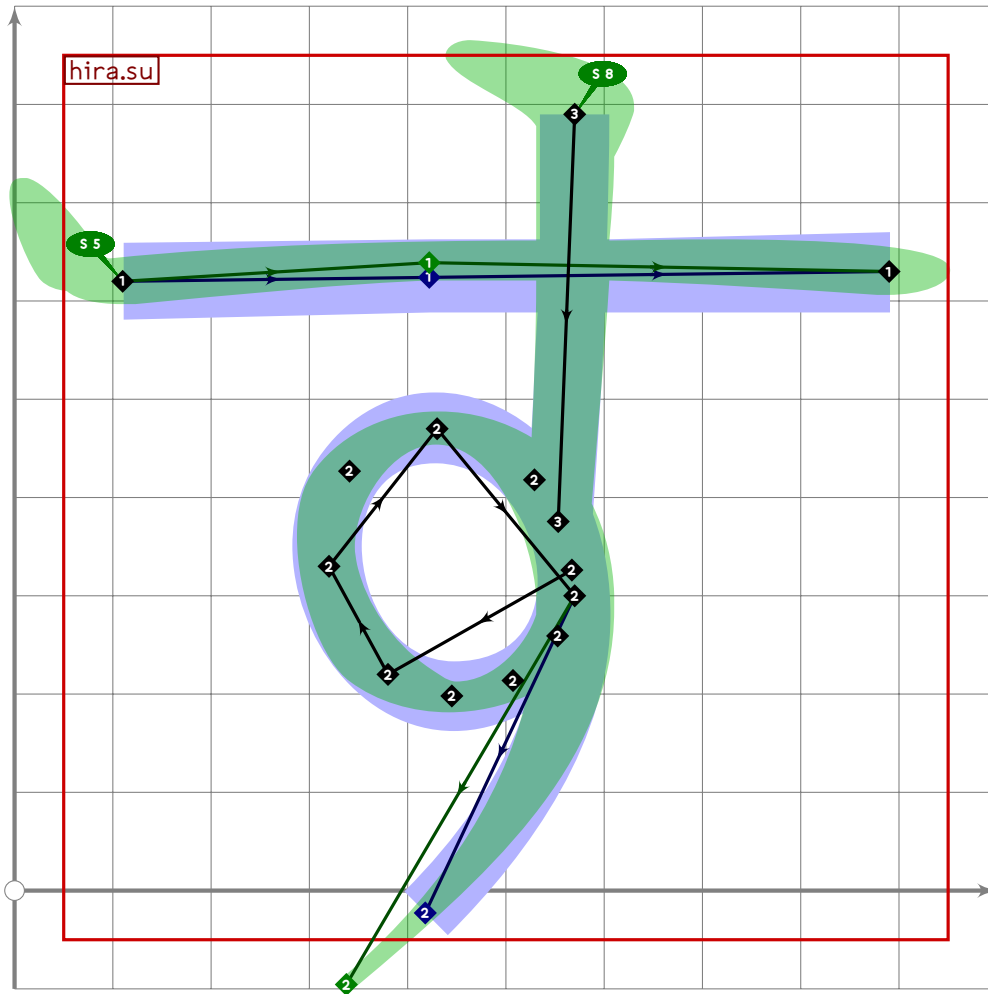


```

233
234 vardef hira.shi =
235   push_pbox_toexpand("hira.shi");
236
237   push_stroke((300,750){down}..tension 2.5..(280,160)..
238     (600,20)..tension 1.5..{curl 0}(990,400),
239     (1.7,1.7)..(1.6,1.6)-(1.5,1.5)-(0.4,0.55));
240   set_boserif(0,0,5);
241   expand_pbox;
242 enddef;

```

**HIRA**

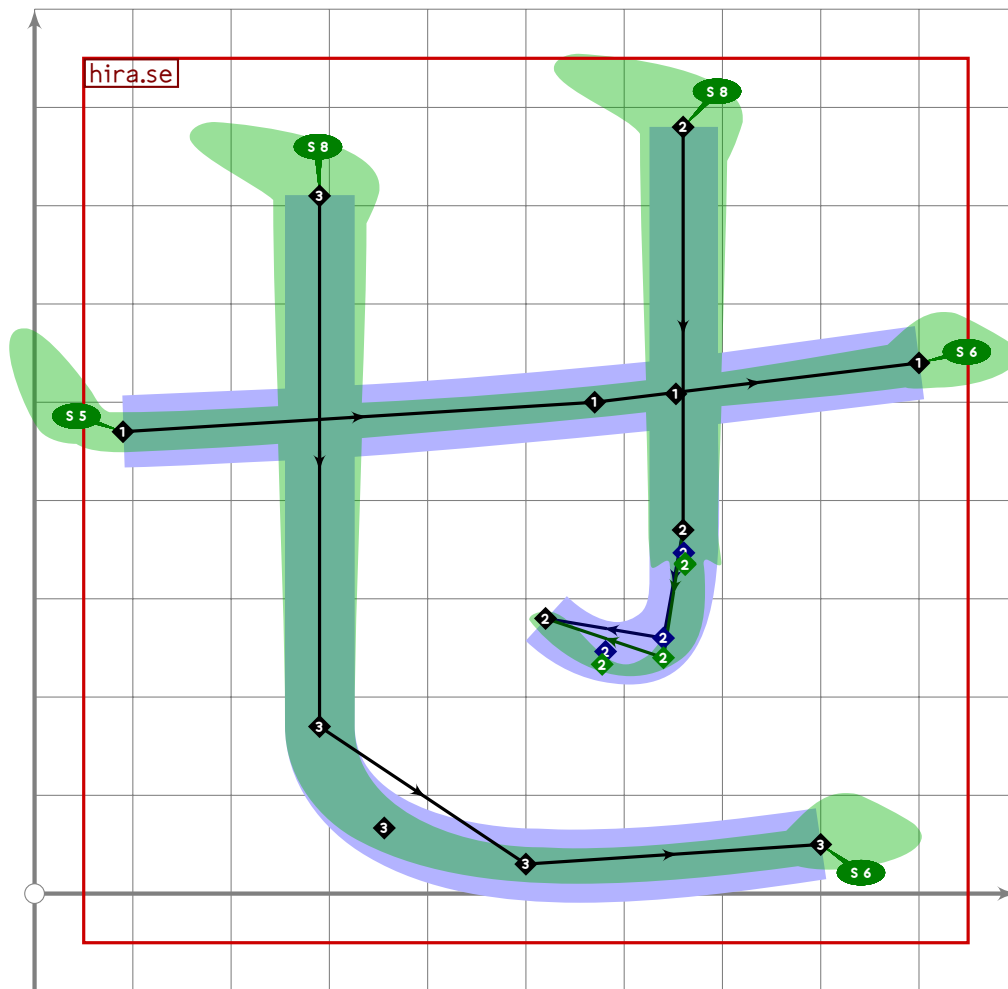


```

243
244 vardef hira.su =
245   push_pbox_toexpand("hira.su");
246
247   push_stroke((110,620)..(0.4[(110,620),(890,630)]+15*up*mincho)..(890,630),
248     (2.2,2.2)-(1.9,1.9)-(2.2,2.2));
249   set_boserif(0,0,5);
250
251   push_stroke((320,330)..(430,470)..(570,300)..{curl 0}(270,-150),
252     (1.3,1.3)-(1.7,1.7)-(1.3,1.3)-(1.7,1.7)-(1.6,1.6)-(0.7,0.7));
253   replace_strokep(0)((point 1.9 of oldp)..(380,220)..oldp);
254
255   push_stroke((570,790){down}..(point 3.7 of get_strokep(0)),
256     (1.6,1.6)..(1.4,1.4));
257   set_boserif(0,0,8);
258   expand_pbox;
259 enddef;

```

**HIRA**

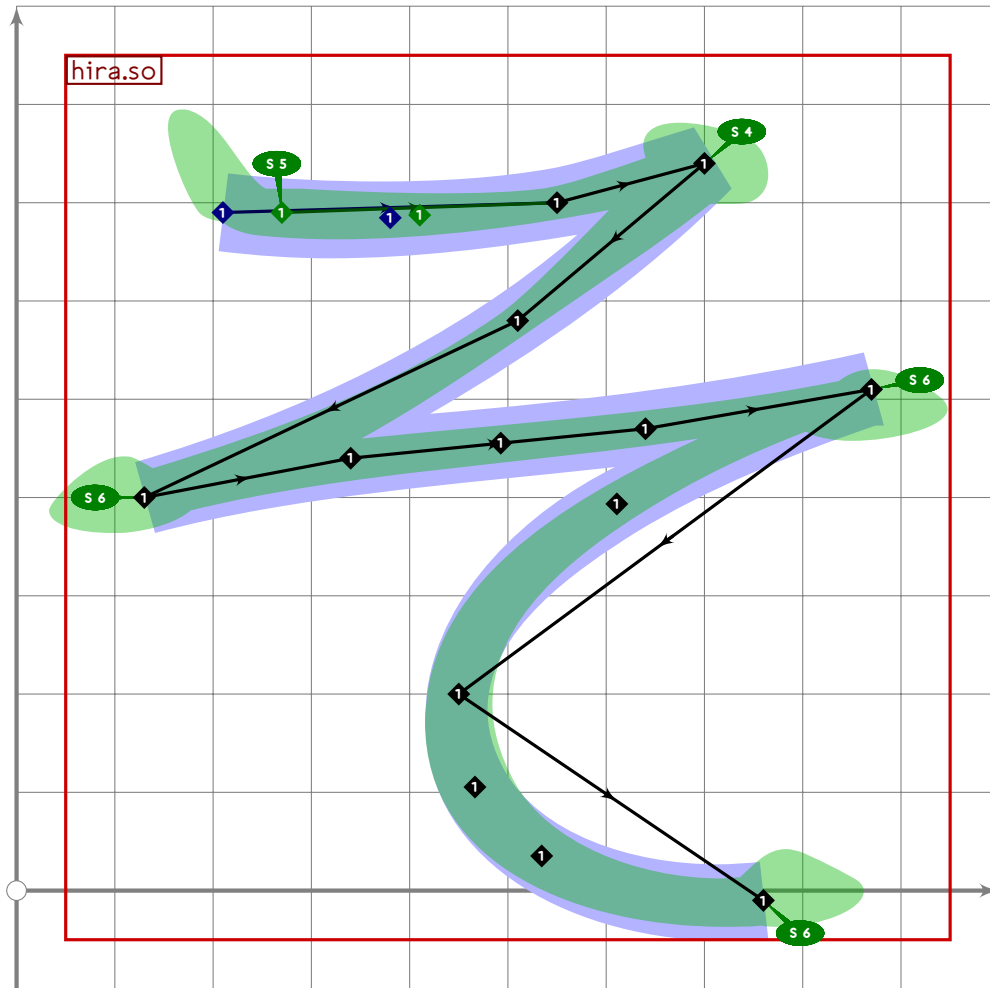


```

260
261 vardef hira.se =
262   push_pbox_toexpand("hira.se");
263
264   push_stroke((90,470)..(570,500)..(900,540),
265     (2,2)..(1.6,1.6)..(2,2));
266   set_boserif(0,0,5);
267   set_boserif(0,2,6);
268
269   push_stroke((660,780)..(660,370){down}..(640,260-20*mincho)..
270     {curl 0.2}(520,280),
271     (1.7,1.7)-(1.5,1.5)-(1.5,1.5)-(1,1));
272   set_boserif(0,0,8);
273
274   push_stroke((290,710)..(290,170){down}..(500,30)..
275     {direction infinity of get_stroke(-1)}(800,50),
276     (1.8,1.8)-(1.5,1.5)-(1.7,1.7)-(1.8,1.8));
277   set_boserif(0,0,8);
278   set_boserif(0,3,6);
279   expand_pbox;
280 enddef;

```

**HIRA**



```

281
282 vardef hira.so =
283   push_pbox_toexpand("hira.so");
284
285   push_stroke(
286     (210+60*mincho,690)..tension 1.2..(550,700)..
287     {curl 0}{(700,740){curl 1}..
288     (510,580)..
289     {curl 1}{(130,400){curl 2}..(340,440)..(640,470)..
290     {curl 1}{(870,510){curl 0}..tension 1.2..(450,200)..{curl 0.2}{(760,-10),
291     (2.3,2.3)-(1.7,1.7)-(1.8,1.8)-
292     (1.2,1.2)-
293     (2.1,2.1)-(1.9,1.9)-(1.7,1.7)-(1.5,1.5)-
294     (1.4,1.4)-(2.3,2.3));
295   set_botip(0,2,0);
296   set_botip(0,4,0);
297   set_botip(0,7,0);
298   set_boserif(0,0,5);
299   set_boserif(0,2,4);
300   set_boserif(0,4,6);
301   set_boserif(0,7,6);

```

**HIRA**

```

302 set_boserif(0,9,6);
303 expand_pbox;
304 enddef;
305

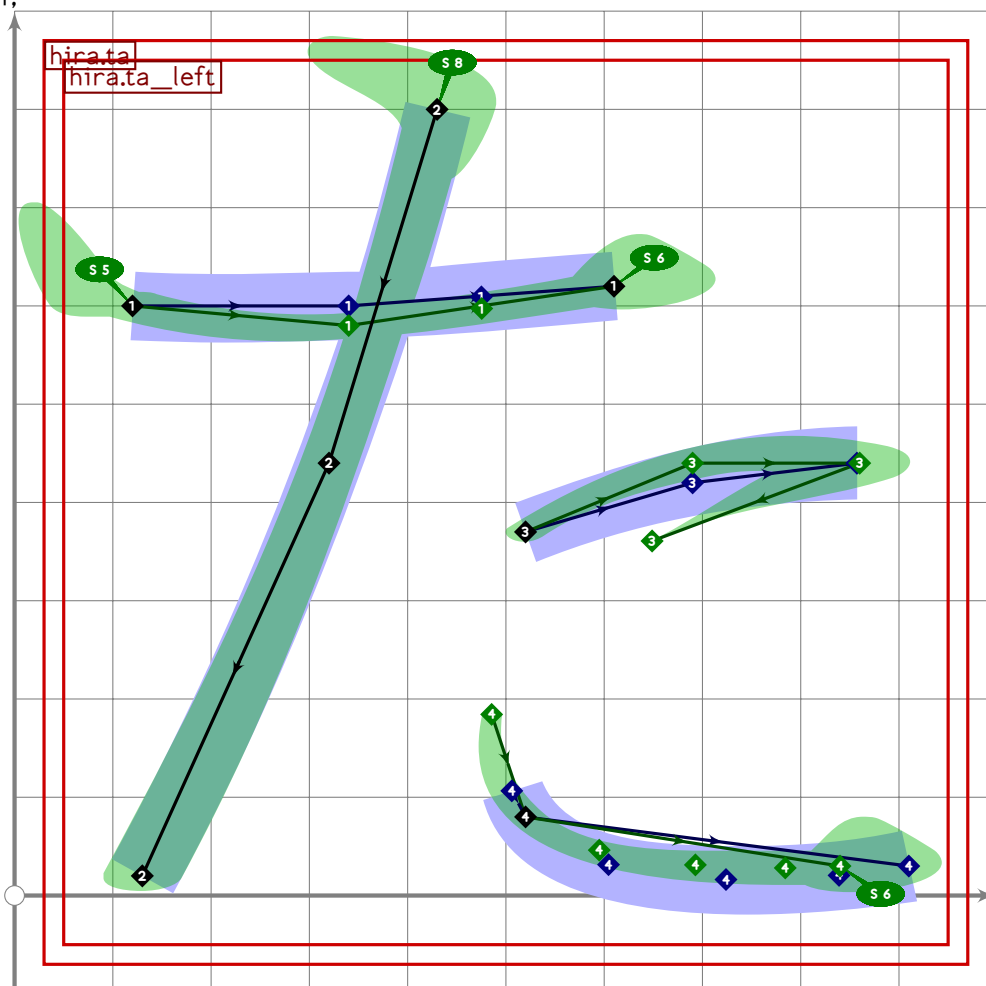
```

## Hiragana Tachitsuteto/Dajizudedo

```

306 %%%%%%%%%% HIRAGANA TACHITSUTETO/DAJIZUDEDO
307
308 vardef hira.ta_left =
309   push_pbox_toexpand("hira.ta_left");
310
311   push_stroke((120,600)..(340,600-20*mincho)..tension 1.5..(610,620),
312     (1.6,1.6)..(1.5,1.5)..(1.7,1.7));
313   set_boserif(0,0,5);
314   set_boserif(0,2,6);
315
316   push_stroke((430,800)..(320,440)..(130,20),
317     (1.4,1.4)..(1.3,1.3)..(1.6,1.6));
318   set_boserif(0,0,8);
319   expand_pbox;
320 enddef;

```

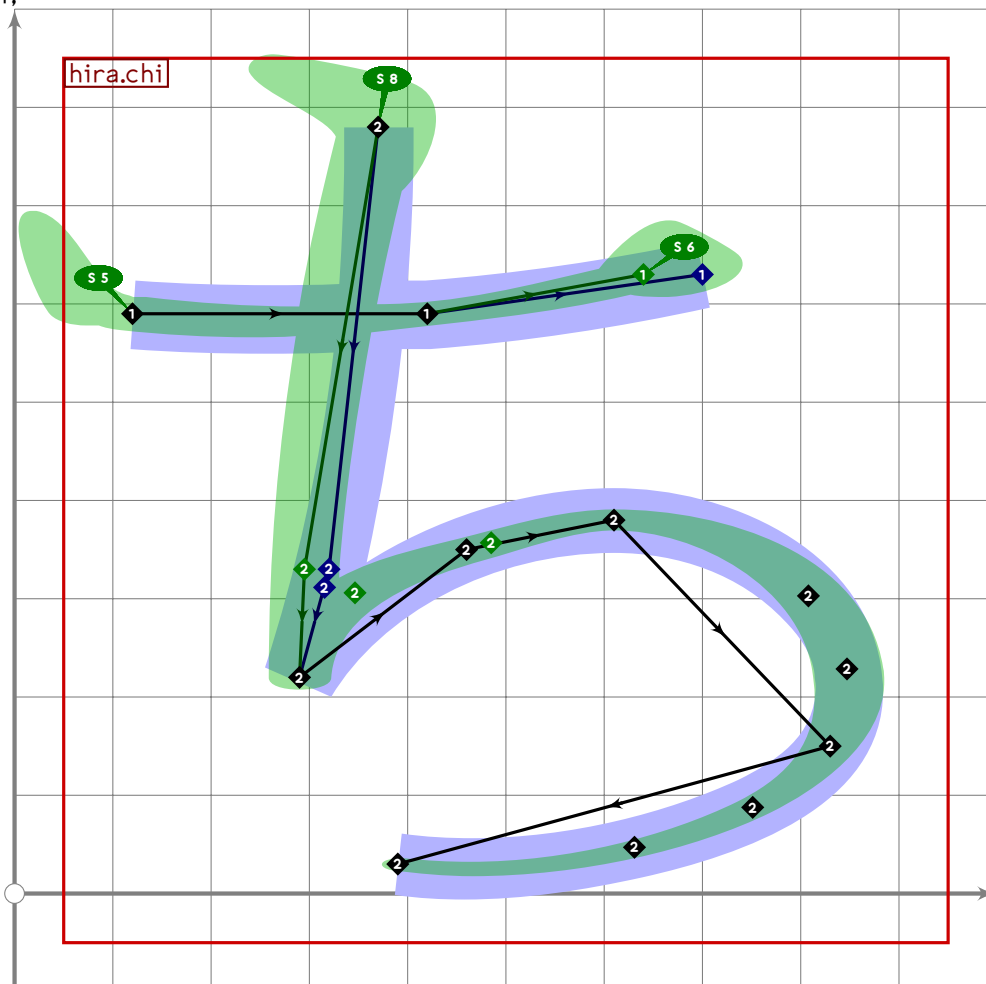


HIRA

```

321
322 vardef hira.ta =
323   push_pbox_toexpand("hira.ta");
324
325   hira.ta_left;
326
327   push_stroke((520,370)..(690,420+20*mincho)..{curl 1.5}(860,440){curl 0}..
328     (610,280+60*mincho)..(520,80)..tension 1.2 and 3..
329     {curl 0.2}(910-70*mincho,30),
330     (1,1,1)-(1.6,1.6)-(2.8,0.99)-(0.45,0.35)-(1,1,1)-(1.9,1.9));
331   set_botip(0,2,0);
332   set_boserif(0,5,6);
333   expand_pbox;
334 enddef;
335
336 vardef hira.chi_bottom =
337   replace_strokep(0)((oldp){-direction infinity of oldp xscaled 2}..
338     (460,350)..(610,380){right}(830,150)..
339     tension 1.4..{curl 0.3}(390,30));
340   replace_strokeq(0)((oldq)..(1.3,1.3)..(1.5,1.5)..(1.5,1.5)..(1,1));
341 enddef;

```



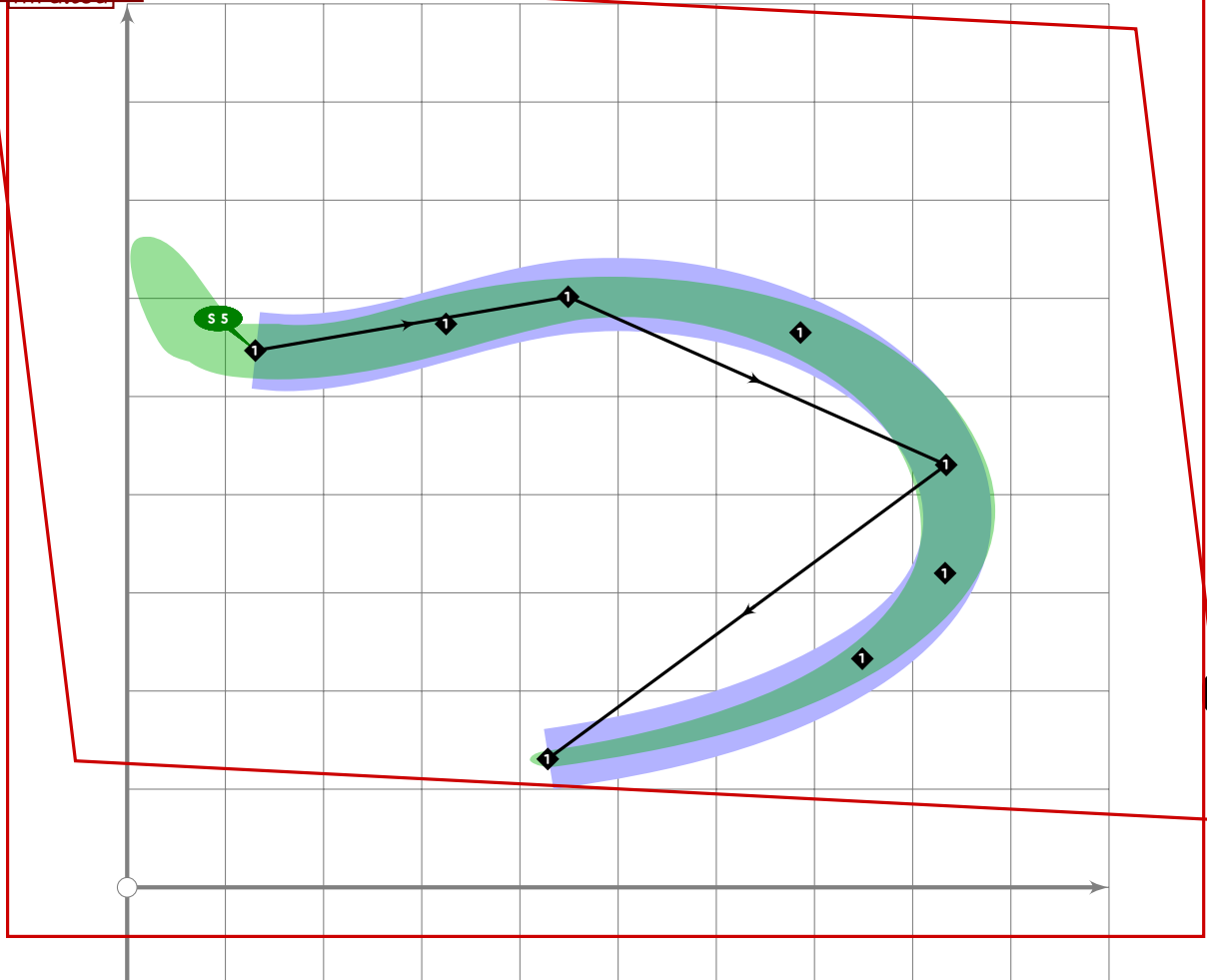
HIRA

```

342
343 vardef hira.chi =
344   push_pbox_toexpand("hira.chi");
345
346   push_stroke((120,590)..(420,590)..(700-60*mincho,630),
347     (1.7,1.7)..(1.5,1.5)..(1.6,1.6));
348   set_boserif(0,0,5);
349   set_boserif(0,2,6);
350
351   push_stroke((370,780)..(320-25*mincho,330)..(290,220),
352     (1.6,1.6)..(1.4,1.4)..(1.5,1.5));
353   hira.chi_bottom;
354   set_botip(0,2,0);
355   set_boserif(0,0,8);
356   expand_pbox;
357 enddef;

```

hira.chi



HIRA

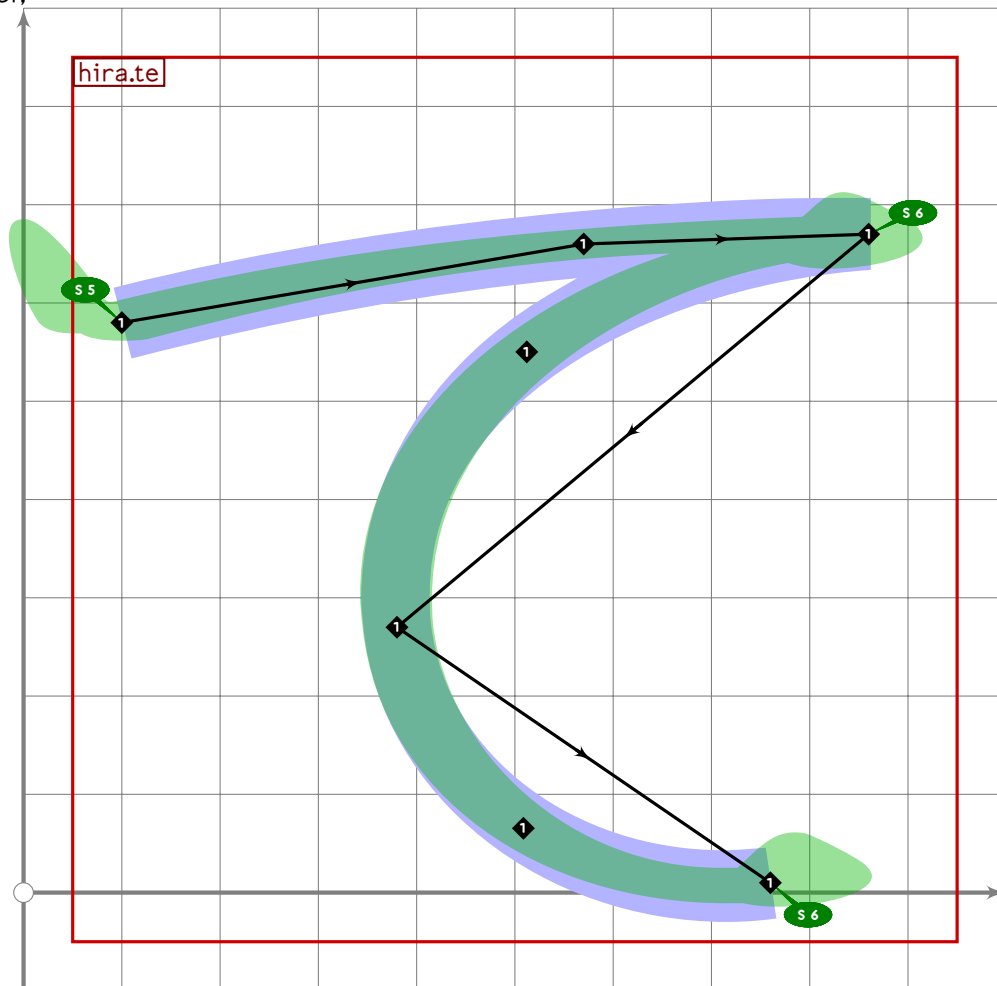
```

358
359 vardef hira.tsu =
360   push_pbox_toexpand("hira.tsu");
361

```

U+3066  
tsuku.uni3066

```
362 begingroup
363   save xf;
364   transform xf;
365   (300,450) transformed xf=(220,560);
366   (750,350) transformed xf=(820,440);
367   (400,0) transformed xf=(400,150);
368   tsu_xform(xf)(hira.u_bowl);
369   set_bosize(0)(100+10*mincho);
370 endgroup;
371 expand_pbox;
372 enddef;
```



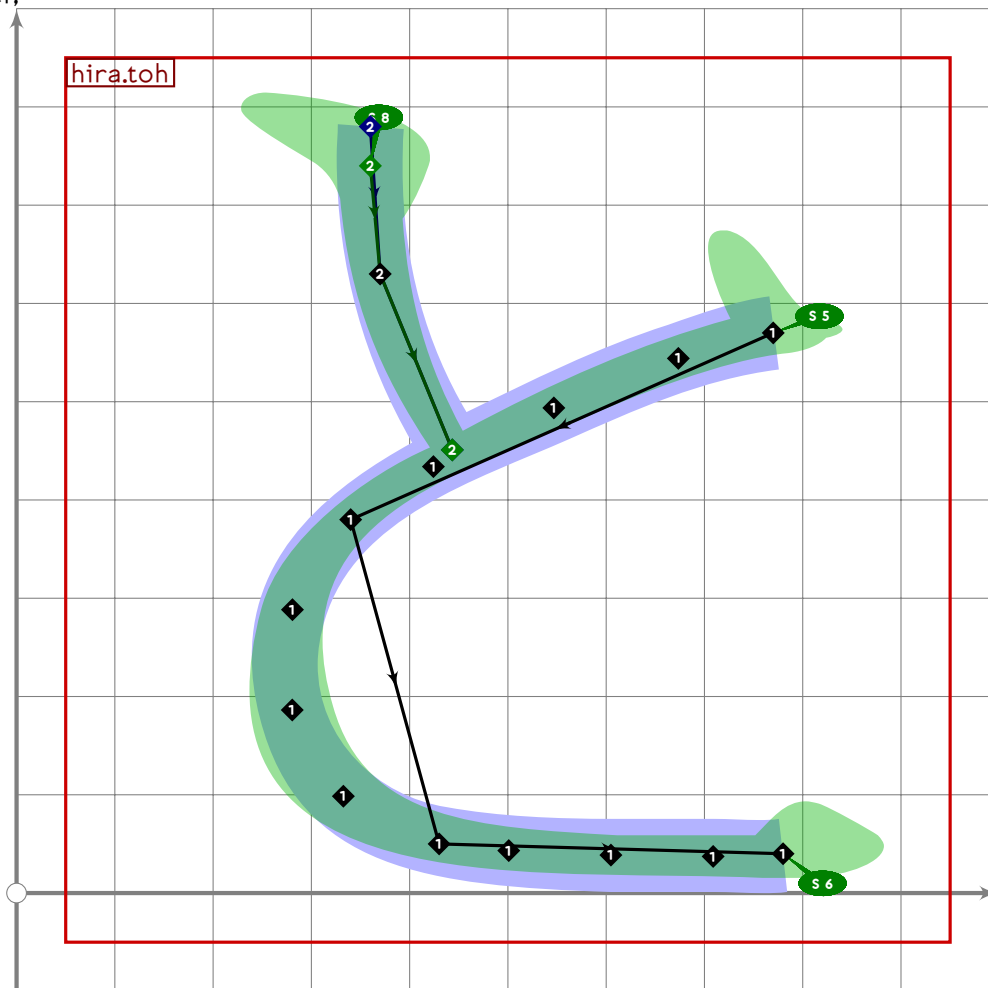
HIRA

```
373
374 vardef hira.te =
375   push_pbox_toexpand("hira.te");
376
377   push_stroke((100,580)..(570,660){curl 1}(860,670){curl 0.2}..
378     (380,270){curl 0.6}(760,10),
379     (19,19)-(1.5,1.5)-(1.8,1.8)-(1.5,1.5)-(1.8,1.8));
380   set_botip(0,2,0);
381   set_boserif(0,0,5);
382   set_boserif(0,2,6);
```

```

383 set_boserif(0,4,6);
384 expand_pbox;
385 enddef;

```



```

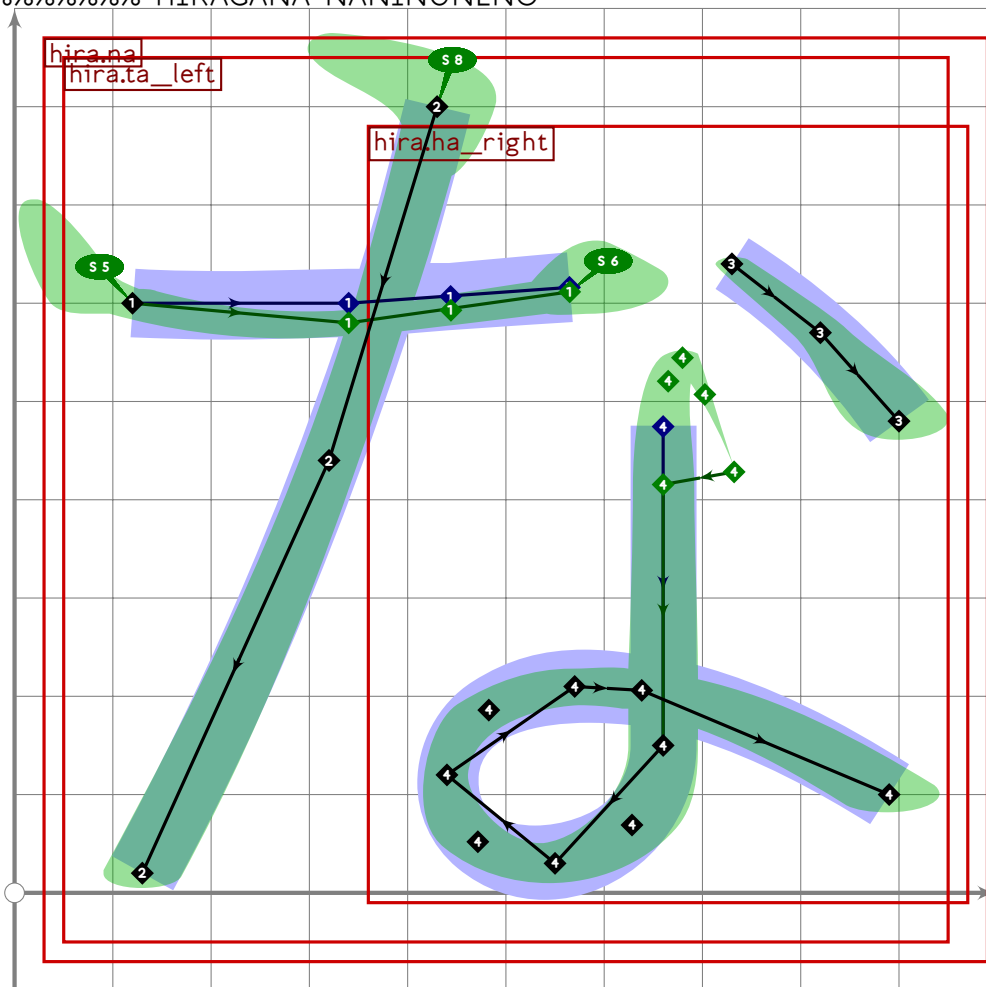
386
387 vardef hira.toh =
388   push_pbox_toexpand("hira.toh");
389
390   push_stroke((770,570)..tension 1.7..(340,380)..(430,50)..
391     tension 2..(780,40),
392     (2,2)-(1.2,1.2)-(2.1,2.1)-(2,2));
393   set_boserif(0,0,5);
394   set_boserif(0,3,6);
395
396   push_stroke(subpath (0,1,97) of
397     ((360,780-40*mincho)..(370,630)..(point 0.7 of get_stroke(0))),
398     (1.4,1.4)-(1.3,1.3)-(1.1,1.1));
399   set_boserif(0,0,8);
400   expand_pbox;
401 enddef;
402

```

**HIRA**

# Hiragana Naninuneno

403 %%%%%%%%% HIRAGANA NANINUNENO



```

404
405 vardef hira.na =
406   push_pbox_toexpand("hira.na");
407
408   hira.ta_left;
409   replace_stroke(-1)(subpath (0,1.8) of oldp);
410
411   push_stroke((730,640)..(820,570)..(900,480),
412     (1,1)..(1.3,1.3)..(1.9,1.9));
413
414   hira.ha_right;
415
416   replace_stroke(0)((point infinity of oldp){-direction infinity of oldp}
417     ..(subpath (0.45+0.1*mincho,infinity) of oldp));
418   replace_stroke(0)((0,-0.4)-(1.7,1)-(1.5,1.5)-(2,2)-
419     (1.1,1.1)-(1.9,1.9));
420   expand_pbox;
421 enddef;
422

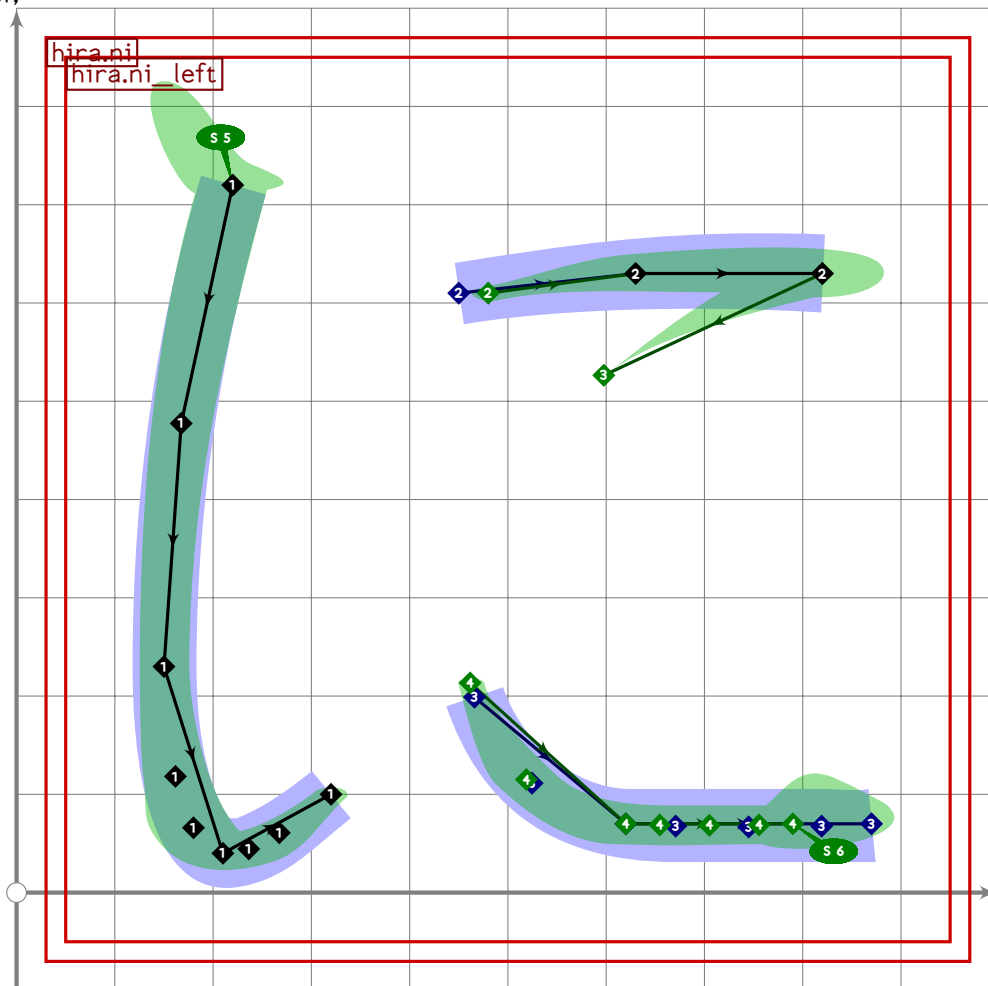
```

**HIRA**

```

423 vardef hira.ni_left =
424   push_pbox_toexpand("hira.ni_left");
425
426   push_stroke((220,720)..(150,230){down}{tension 1.5..(210,40)..
427     tension 1.5..{curl 0}(320,100),
428     (1.5,1.5)..(1.2,1.2)..(1.8,1.8)..(1,1));
429   replace_strokep(0)(insert_nodes(oldp)(0.5));
430   replace_strokeq(0)(insert_nodes(oldq)(0.5));
431   set_boserif(0,0,5);
432   expand_pbox;
433 enddef;

```



HIRA

```

434
435 vardef hira.ni =
436   push_pbox_toexpand("hira.ni");
437
438   hira.ni_left;
439
440   push_stroke((450+30*mincho,610)..(630,630)..(820,630),
441     (1,1)..(1.9,1.9)..(2.2,2.2));
442
443   push_stroke((point infinity of get_strokep(0)){curl 0.6}..(530,460)..

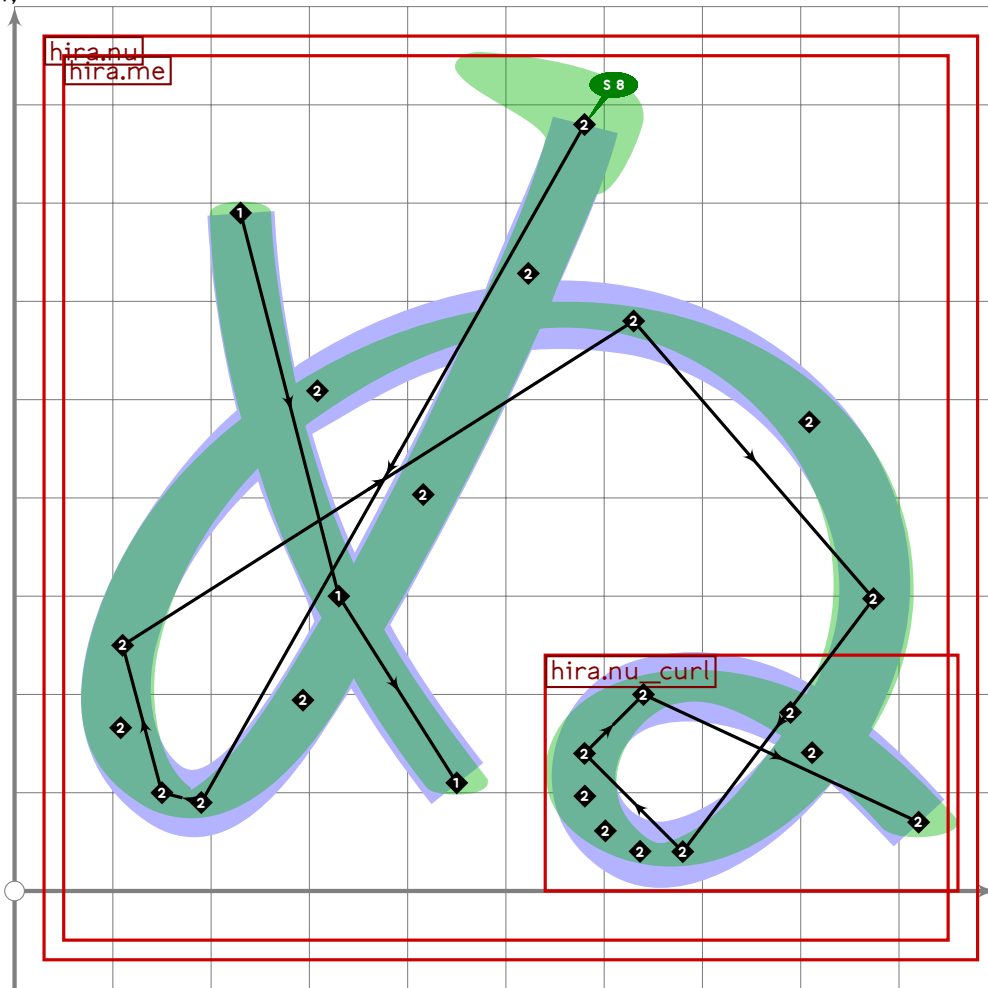
```

U+306C  
tsuku.uni306C

```

444      (460,220)..(620,70)..tension 2..(870-80*mincho,70),
445      (3,0.7)-(0,0)-(0.8,0.9)-(2,2)-(1.9,1.9));
446      set_boserif(0.4,6);
447      expand_pbox;
448  enddef;
449
450  vardef hira.nu_curl =
451    begingroup
452      push_pbox_explicit("hira.nu_curl",
453        identity xyscaled (420,240) shifted (540,0));
454      (680,40)..(580,140)..(640,200)..{curl 0}(920,70)
455    endgroup
456  enddef;

```



**HIRA**

```

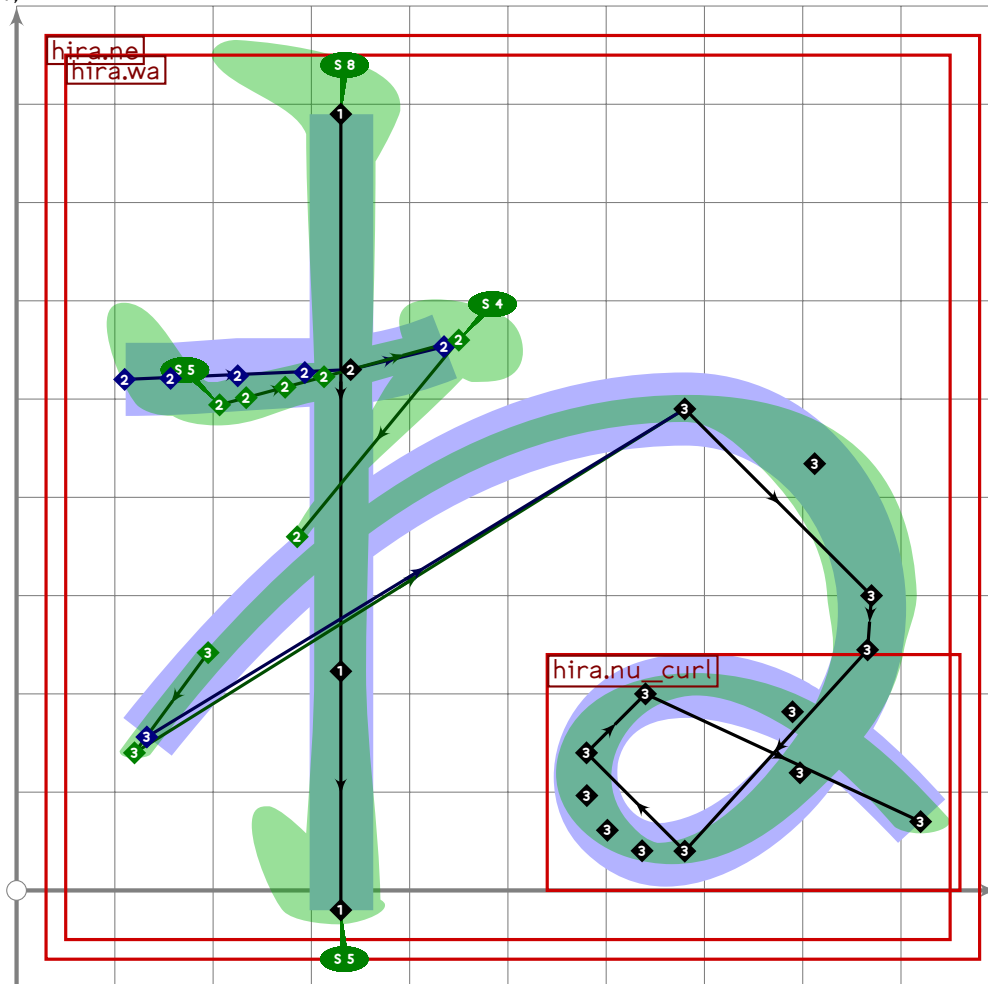
457
458  vardef hira.nu =
459    push_pbox_toexpand("hira.nu");
460
461    hira.me;
462
463    replace_strokep(0)((subpath (0,4.8) of oldp)..tension 1.2..
464      hira.nu_curl);

```

```

465 replace_strokeq(0)((1.5,1.5)–(1.4,1.4)–(1.6,1.6)–(1.4,1.4)–
466 (1.6,1.6)–(1.6,1.6)–(1.7,1.7)–(1.3,1.3)–(1.6,1.6));
467 set_boserif(0,0,8);
468 expand_pbox;
469 enddef;

```

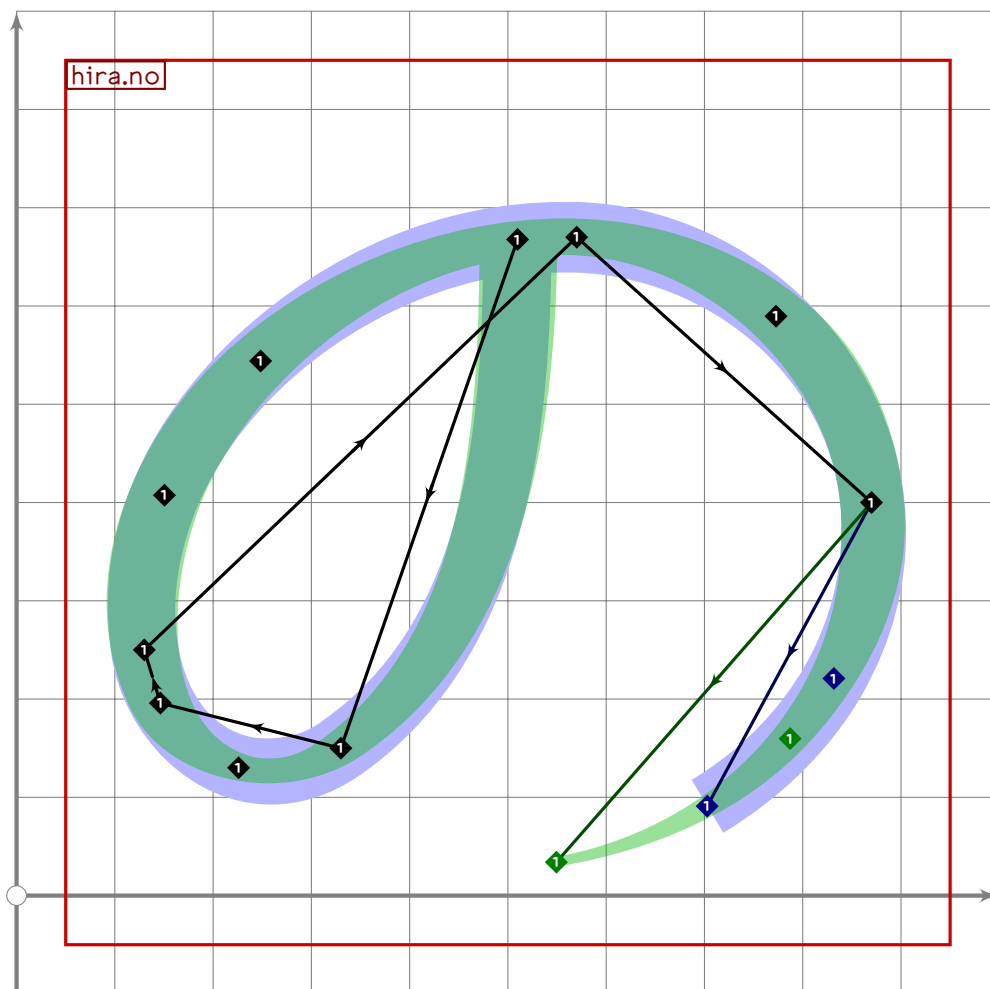


```

470
471 vardef hira.ne =
472   push_pbox_toexpand("hira.ne");
473
474   hira.wa;
475
476   replace_strokeq(0)((subpath (0,6.1) of oldp)..tension 1.2..
477     hira.nu_curl);
478   replace_strokeq(0)((2,2)–(1.6,1.6)–(2.2,0.9)–(0.7,0.7)–(0.97,0.97)–
479     (2,2)–(1.5,1.5)–(1.4,1.4)–(1.4,1.4)–(1.2,1.2)–(1.3,1.3));
480   expand_pbox;
481 enddef;

```

**HIRA**



```

482
483 vardef hira.no =
484   push_pbox_toexpand("hira.no");
485
486   begingroup
487     save px,py;
488     path px,py;
489     px:=(410,30)..(130,250)..tension 1.1..(570,670)..(870,400)..cycle;
490     py:=(510,770){down}{dir 215}{330,150};
491
492     px:=subpath (0.85,4) of px;
493     push_stroke(
494       (subpath (xpart (py intersectiontimes px),infinity) of py)..px,
495       (1.6,1.6)-(1.3,1.3)-(1.4,1.4)-(1.5,1.5)-(1.8,1.8)-
496       (1.4,1.4)-(0.70,7));
497   endgroup;
498   expand_pbox;
499 enddef;
500

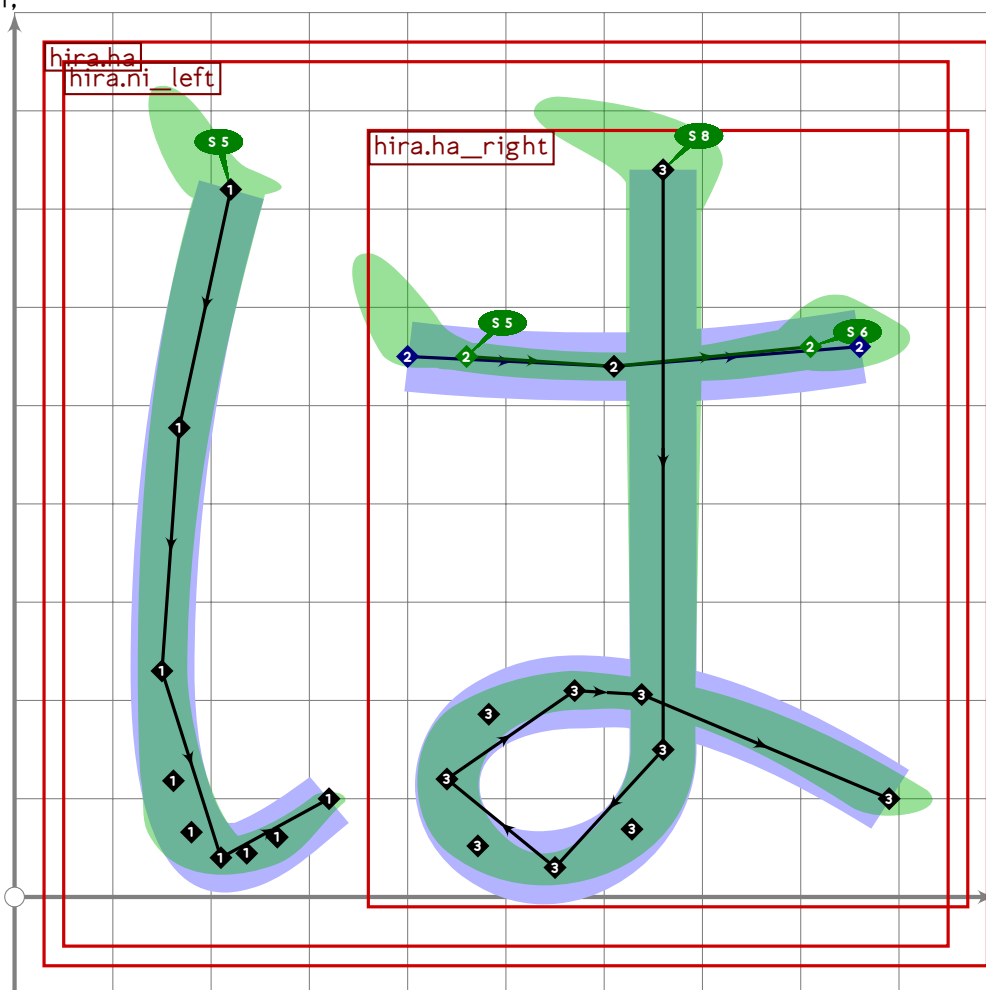
```

# Hiragana Hahifuheho/Babibubebo/Papipupepo

```

501 %%%%%%%%%% HIRAGANA HAHIFUHEHO/BABIBUBEBO/PAPIPUPEPO
502
503 vardef hira.ha_right =
504   push_pbox_explicit("hira.ha_right",
505     identity xyscaled (610,790) shifted (360,-10));
506
507   push_stroke(insert_nodes((660,740)..(660,150){down}..(550,30)..(440,120)..
508     (570,210)..{curl 0.17}(890,100))(4.2),
509     (1.6,1.6)–(1.4,1.4)–(1.6,1.6)–(1.3,1.3)–
510     (2,2)–(1.6,1.6)–(1.7,1.7));
511 enddef;

```



HIRA

```

512
513 vardef hira.ha =
514   push_pbox_toexpand("hira.ha");
515
516   hira.ni_left;
517
518   push_stroke((400+60*mincho,550)..(610,540)..(860-50*mincho,560),
519     (1.6,1.6)..(1.6,1.6)..(2,2));

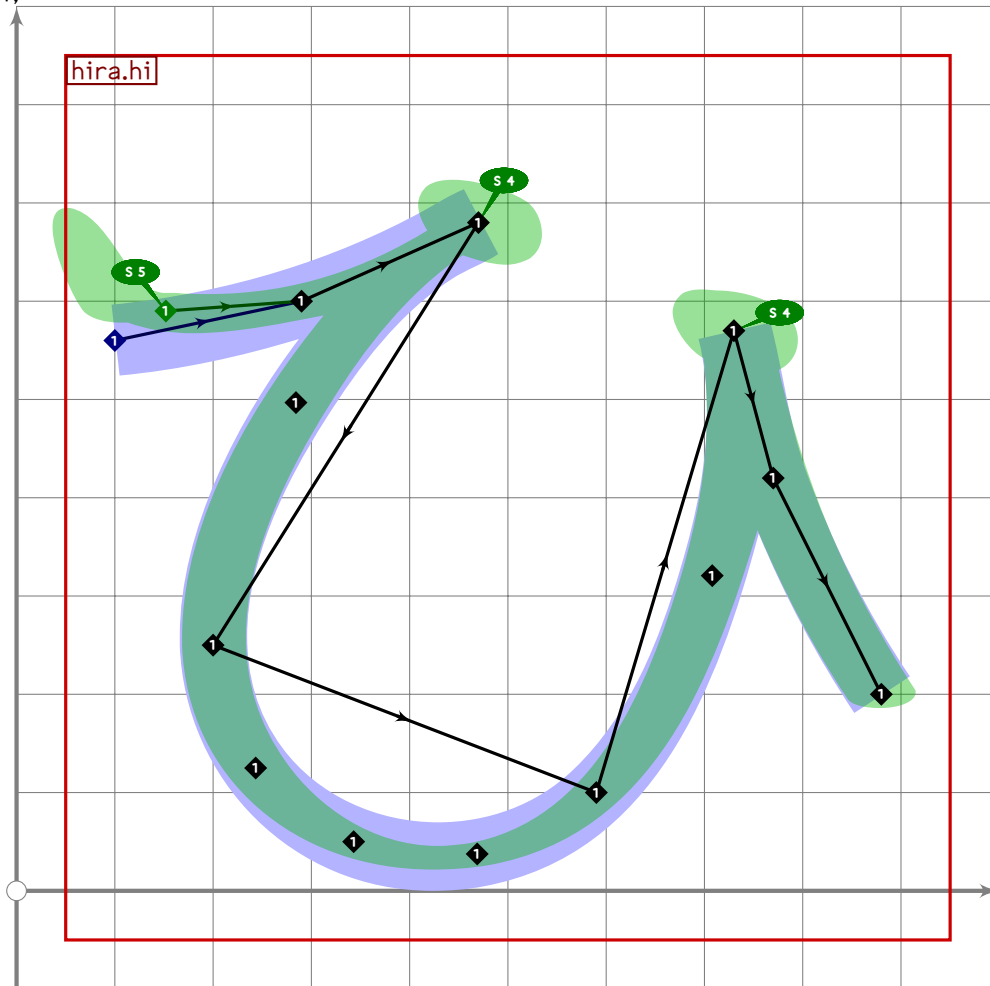
```

U+3072  
tsuku.uni3072

```

520 set_boserif(0,0,5);
521 set_boserif(0,2,6);
522
523 hira.ha_right;
524 set_boserif(0,0,8);
525 expand_pbox;
526 enddef;

```



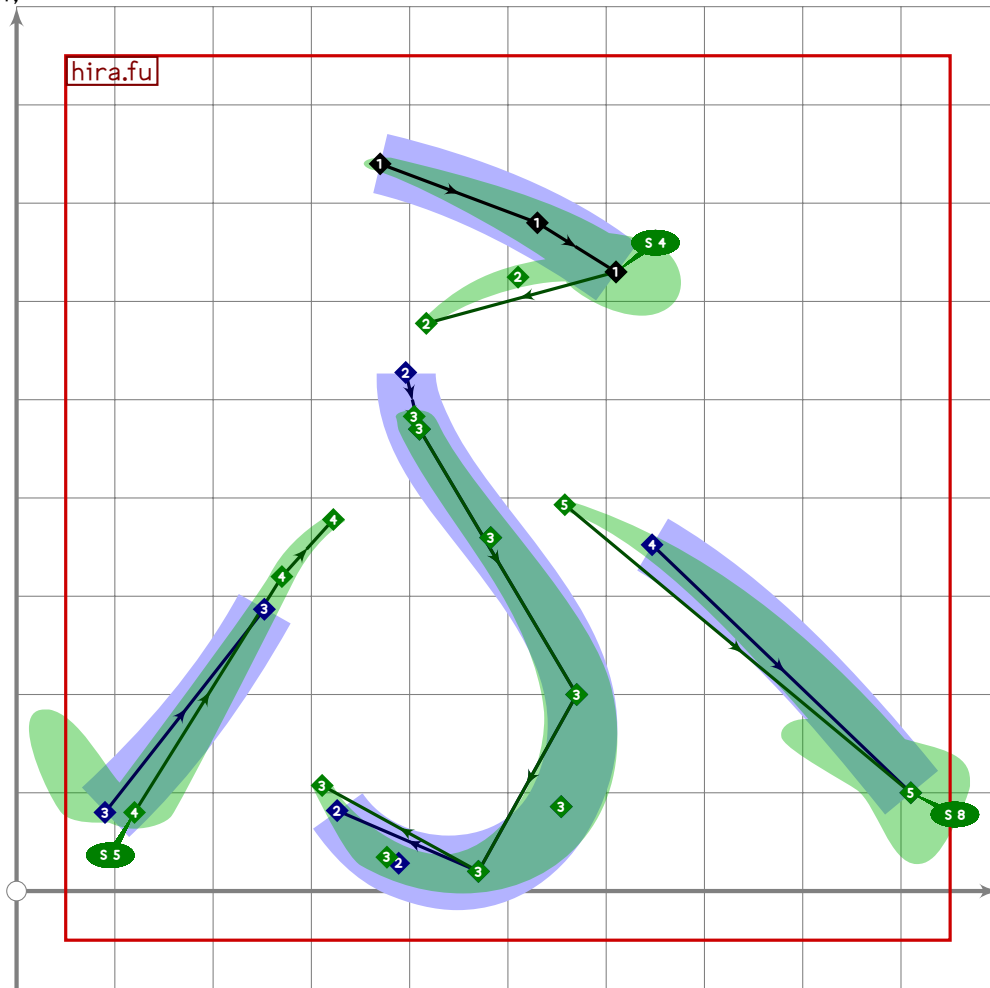
```

527
528 vardef hira.hi =
529   push_pbox_toexpand("hira.hi");
530
531   push_stroke(((100,560)+60*mincho*dir 30)..(290,600)..
532     {curl 1}(470,680){curl 1}..
533     tension 1.3..(200,250)..(590,100)..tension 1.3..
534     {curl 1}(730,570){curl 1}..(770,420)..(880,200),
535     (1,8,1,8)-(1,7,1,7)-(1,5,1,5)-
536     (1,4,1,4)-(1,4,1,4)-(1,4,1,4)-(1,3,1,3)-(1,5,1,5));
537   set_botip(0,2,0);
538   set_botip(0,5,0);
539   set_boserif(0,0,5);
540   set_boserif(0,2,4);

```

**HIRA**

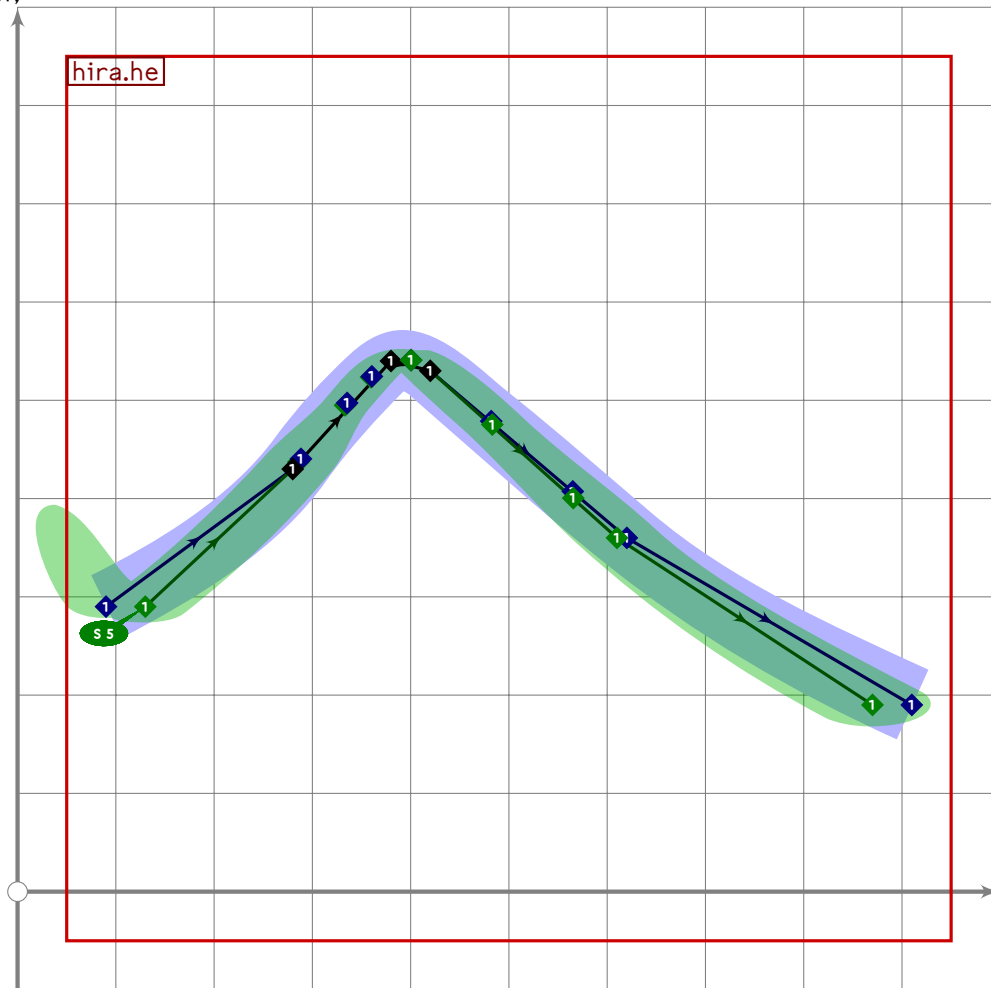
```
541 set_boserif(0,5,4);
542 expand_pbox;
543 enddef;
```



```
544
545 vardef hira.fu =
546   push_pbox_toexpand("hira.fu");
547
548   push_stroke((370,740)..(530,680)..(610,630),
549     (1,1)-(1.7,1.7)-(1.8,1.8));
550   set_boserif(0,2,4);
551
552   push_stroke((610,630)..tension 14..(410,570)..(410,470)..
553     (570,200)..(470,20)..{curl 0.3}(290,270),
554     (2.6,0.79)-(0.72,0.72)-(0.85,1.35)-(1.5,1.5)-(2,2)-(0,0));
555
556   push_stroke((90+30*mincho,80)..(270,320){dir 63}..(480,410)..
557     {curl 0}(910,100),
558     (1.4,1.7)-(0.9,0.9)-(0.7,0.7)-(1.6,1.6));
559   set_boserif(0,0,5);
560   set_boserif(0,3,8);
561   expand_pbox;
```

**HIRA**

562 enddef;



563

564 vardef hira.he =

565 push\_pbox\_toexpand("hira.he");

566

567 push\_stroke((90+40\*mincho,290){curl 0.2}..(280,430)..tension 2..(380,540)..

568 (420,530)..tension 2..(620-10\*mincho,360)..{curl 0.2}(910-40\*mincho,190),

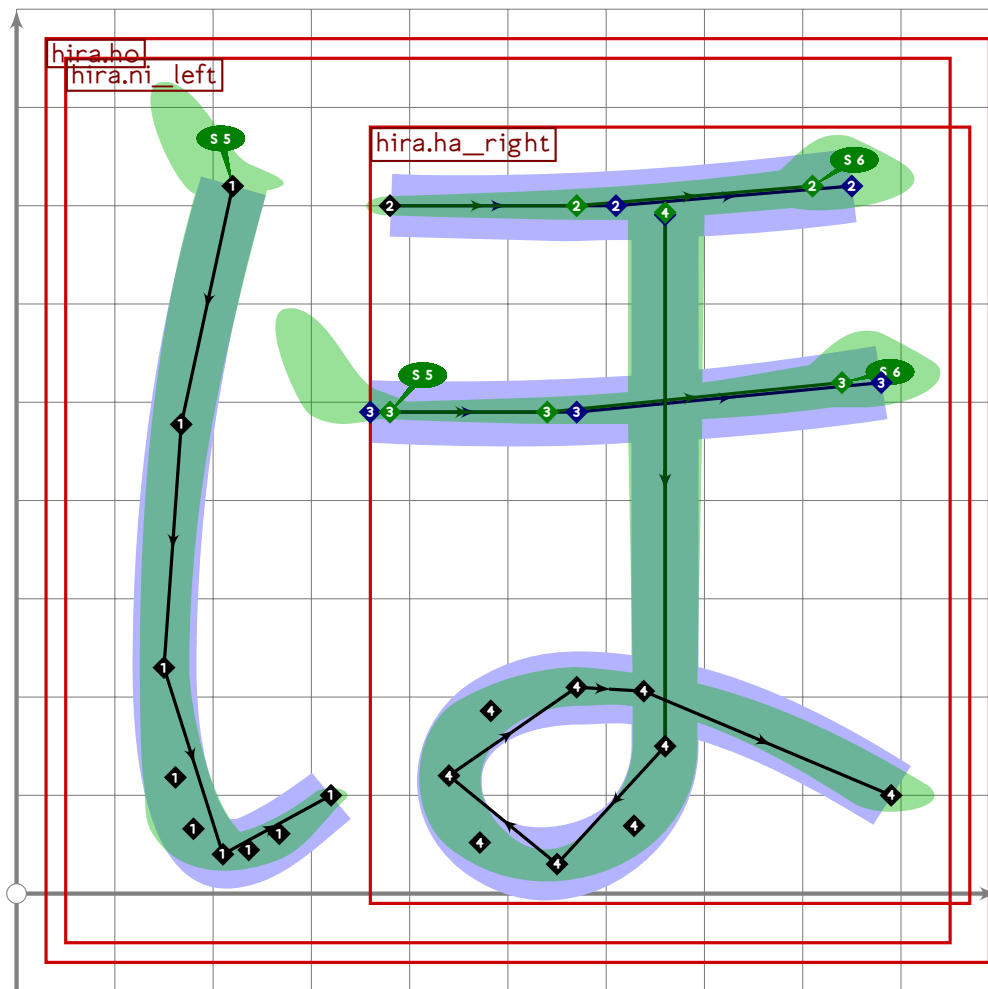
569 (1.7,1.7)..(1.6,1.6)..(1.2,1.2)..(1.3,1.3)..(1.7,1.7)..(2.1,2.1));

570 set\_boserif(0,0,5);

571 expand\_pbox;

572 enddef;

**HIRA**



```

573
574 vardef hira.ho =
575   push_pbox_toexpand("hira.ho");
576
577   hira.ni_left;
578
579   push_stroke((380,700)..(610-40*mincho,700)..(850-40*mincho,720),
580     (1,2,1,2)..(1,6,1,6)..(1,9,1,9));
581   set_boserif(0,2,6);
582
583   push_stroke((360+20*mincho,490)..(570-30*mincho,490)..(880-40*mincho,520),
584     (1,2,1,2)..(1,5,1,5)..(2,2));
585   set_boserif(0,0,5);
586   set_boserif(0,2,6);
587
588   hira.ha_right;
589   replace_strokep(0)(subpath
590     (xpart (oldp intersectiontimes get_strokep(-2))+0.02,infinity) of oldp);
591   expand_pbox;
592 enddef;
593

```

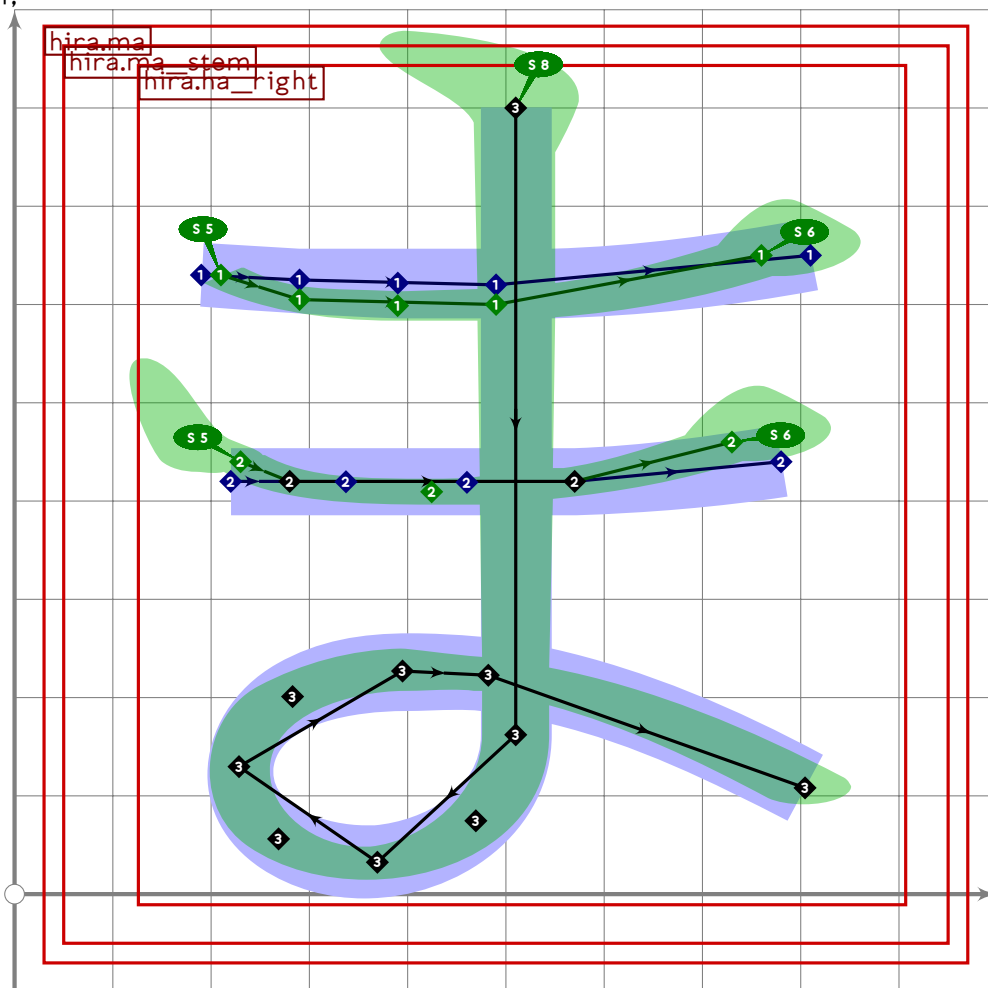
**HIRA**

## Hiragana Mamimumemo

```

594 %%%%%%%%%% HIRAGANA MAMIMUMEMO
595
596 vardef hira.ma_stem =
597   push_pbox_toexpand("hira.ma_stem");
598
599   begingroup
600     transform xf;
601
602     (660,0) transformed xf = (510,0);
603     (660,740) transformed xf = (510,800);
604     (910,0) transformed xf = (830,0);
605
606     tsu_xform(xf)(hira.ha_right);
607     set_boserif(0,0,8);
608   endgroup;
609   expand_pbox;
610 enddef;

```



HIRA

```

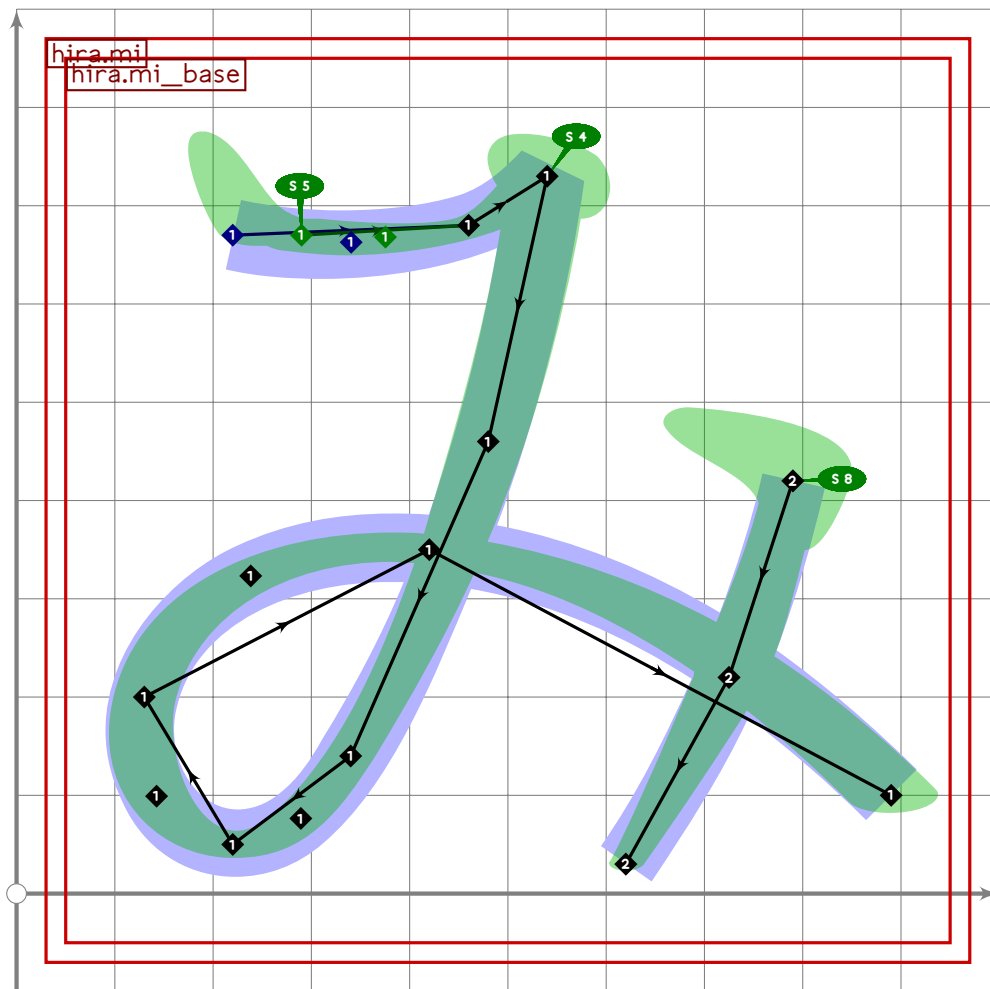
611
612 vardef hira.ma =

```

```

613 push_pbox_toexpand("hira.ma");
614
615 push_stroke((190+20*mincho,630)..(290,625-20*mincho)..tension 1.3..
616 (490,620-20*mincho)..(810-50*mincho,650),
617 (1.3,1.3)-(1.4,1.4)-(1.7,1.7)-(1.9,1.9));
618 set_boserif(0,0,5);
619 set_boserif(0,3,6);
620
621 push_stroke((220+10*mincho,420+20*mincho)..(280,420)..tension 1.3..
622 (570,420)..(780-50*mincho,440+20*mincho),
623 (1.3,1.3)-(1.3,1.3)-(1.6,1.6)-(1.8,1.8));
624 set_boserif(0,0,5);
625 set_boserif(0,3,6);
626
627 hira.ma_stem;
628 expand_pbox;
629 enddef;
630
631 vardef hira.mi_base =
632 push_pbox_toexpand("hira.mi_base");
633
634 push_stroke(
635 (220+70*mincho,670)..tension 1.3..(460,680)..
636 {curl 1}(540,730){curl 1}..
637 (480,460)..(340,140)..(220,50)..(130,200)..(420,350)..
638 {curl 0.4}(890,100),
639 (1.7,1.7)-(1.3,1.3)-(1.6,1.6)-
640 (1.5,1.5)-(1.2,1.2)-(1.5,1.5)-(1.4,1.4)-(1.6,1.6)-
641 (1.8,1.8));
642 set_botip(0,2,0);
643 set_boserif(0,0,5);
644 set_boserif(0,2,4);
645 expand_pbox;
646 enddef;

```

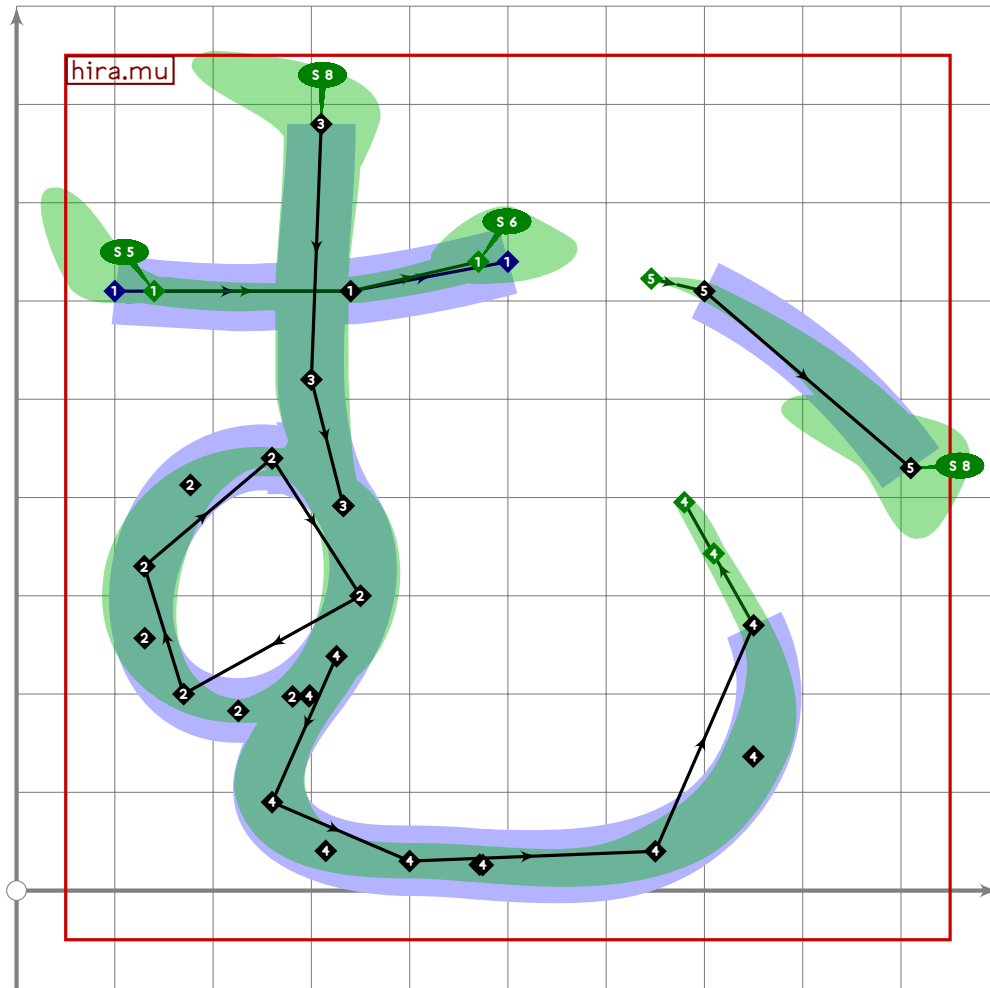


```

647
648 vardef hira.mi =
649   push_pbox_toexpand("hira.mi");
650
651   hira.mi_base;
652
653   push_stroke((790,420)..(725,220)..(620,30),
654     (1,3,1,3)-(1,5,1,5)-(1,1));
655   set_boserif(0,0,8);
656   expand_pbox;
657 enddef;

```

**HIRA**



```

658
659 vardef hira.mu =
660   push_pbox_toexpand("hira.mu");
661
662   push_stroke((100+40*mincho,610)..(340,610)..(500-30*mincho,640),
663     (1.6,1.6)-(1.4,1.4)-(1.5,1.5));
664   set_boserif(0,0,5);
665   set_boserif(0,2,6);
666
667   push_stroke((260,440)..(350,300)..(170,200)..(130,330)..cycle,
668     (1.3,1.3)..(1.6,1.6)..(1.3,1.3)..(1.6,1.6)..cycle);
669
670   push_stroke((310,780){down}..(300,520)..
671     (point 0.5 of get_stroke(0)){direction 0.5 of get_stroke(0)},
672     (1.6,1.6)-(1.5,1.5)-(1.2,1.2));
673   set_boserif(0,0,8);
674
675   push_stroke(
676     (point 1.25 of get_stroke(-1)){direction 1.20 of get_stroke(-1)}..
677     (260,90)..(400,30){right}..(650,40)..(750,270)..
678     tension 2..(600,590)..(700,610)..(910,430),

```

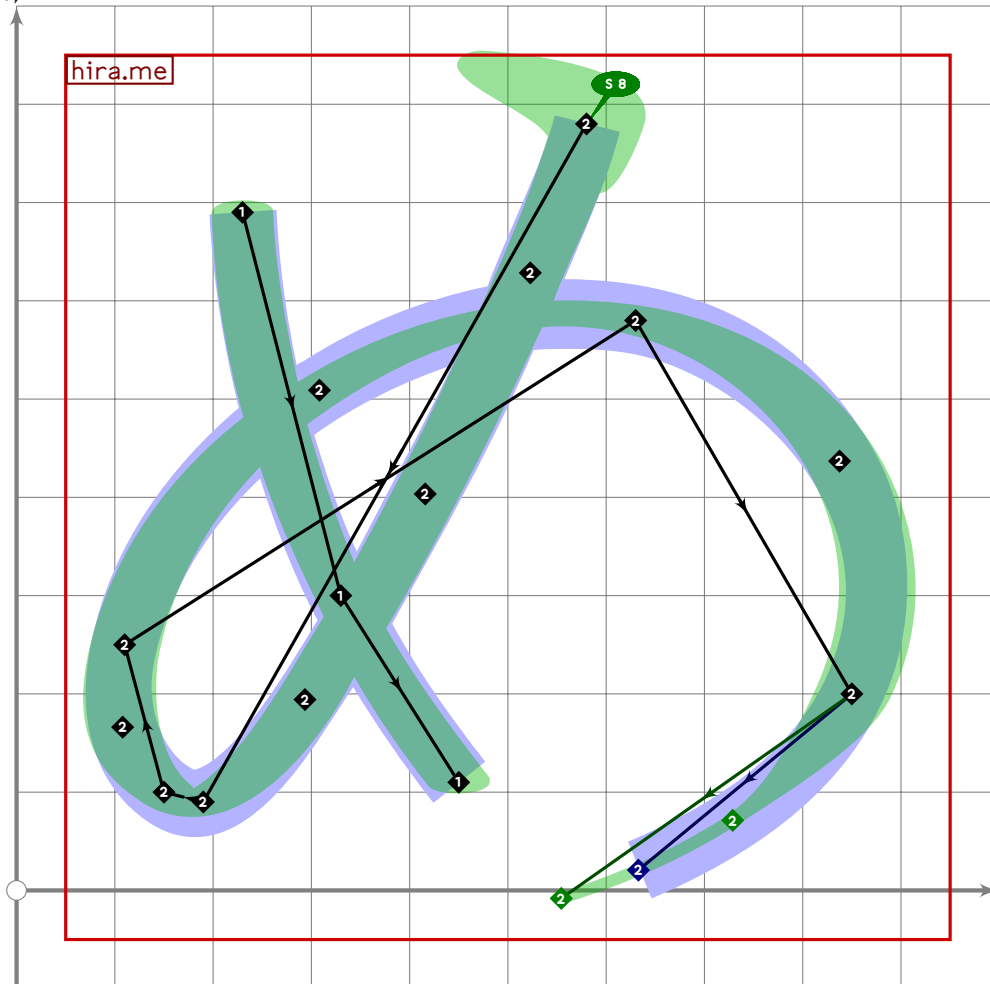
**HIRA**

U+3081  
tsuku.uni3081

```

679 (1,4,1,4)–(1,5,1,5)–(1,8,1,8)–(1,9,1,9)–
680 (1,1)–(0,6,0,6)–(1,1)–(1,6,1,6));
681 set_boserif(0,7,8);
682 expand_pbox;
683 enddef;

```

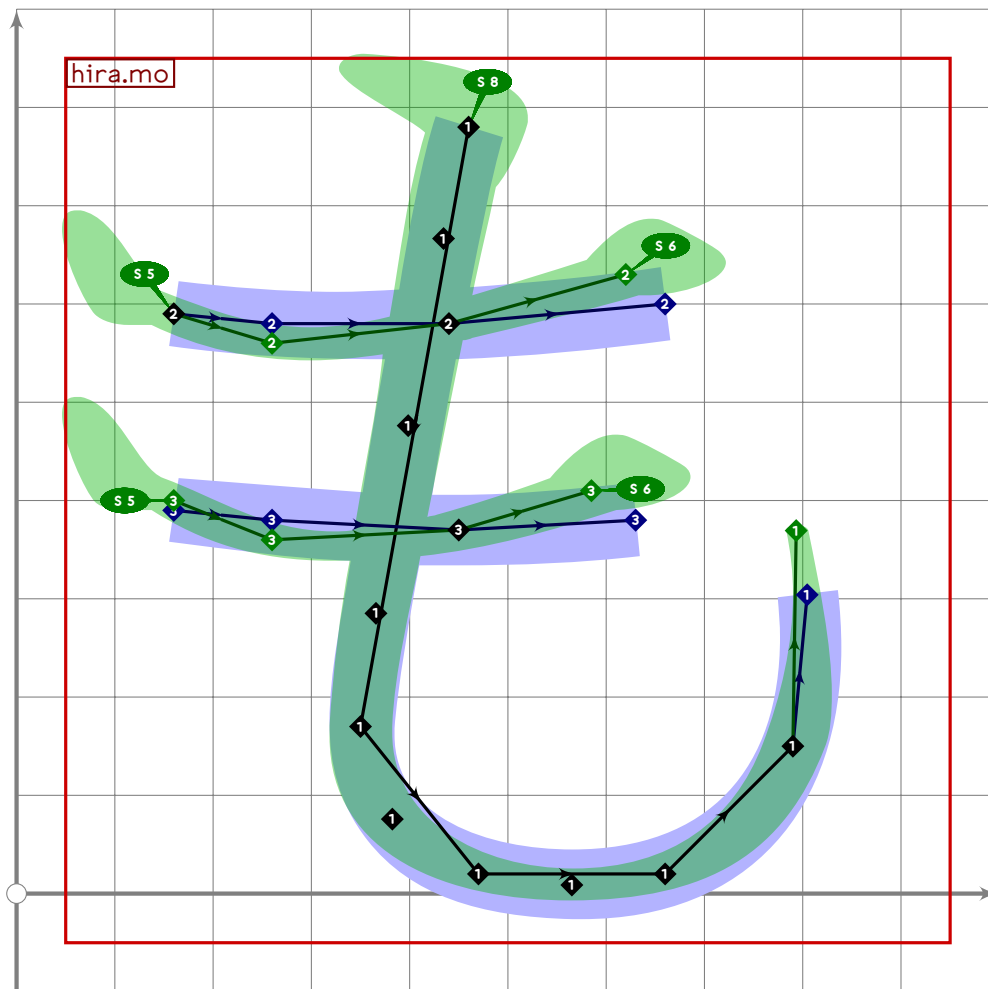


```

684
685 vardef hira.me =
686   push_pbox_toexpand("hira.me");
687
688   push_stroke((230,690)..(330,300)..(450,110),
689     (1,4,1,4)–(1,3,1,3)–(1,4,1,4));
690
691   push_stroke((580,780){curl 0.2}..tension 2.5..(190,90)..
692     (150,100)..(110,250)..tension 1.1..(630,580)..
693     (850,200)..tension 1.1..{curl 0.2}(470,30),
694     (1,5,1,5)–(1,4,1,4)–(1,6,1,6)–(1,4,1,4)–
695     (1,6,1,6)–(1,6,1,6)–(0,6,0,6));
696   set_boserif(0,0,8);
697   expand_pbox;
698 enddef;

```

**HIRA**



```

699
700 vardef hira.mo =
701   push_pbox_toexpand("hira.mo");
702
703   push_stroke((460,780)..tension 3..(350,170)..(470,20)..
704     tension 1.2..(660,20)..(790,150)..{curl 0}{(770,460),
705     (1,7,1,7)-(1,4,1,4)-(1,6,1,6)-(1,7,1,7)-(1,4,1,4)-(0,6,0,6));
706   set_boserif(0,0,8);
707
708   push_stroke((160,590)..(260,580-20*mincho)..(440,580)..
709     (660-40*mincho,600+30*mincho),
710     (1,4,1,4)-(1,6,1,6)-(1,8,1,8)-(2,2));
711   set_boserif(0,0,5);
712   set_boserif(0,3,6);
713
714   push_stroke((160,390+10*mincho)..(260,380-20*mincho)..(450,370)..
715     (630-45*mincho,380+30*mincho),
716     (1,4,1,4)-(1,6,1,6)-(1,8,1,8)-(2,2));
717   set_boserif(0,0,5);
718   set_boserif(0,3,6);
719   expand_pbox;

```

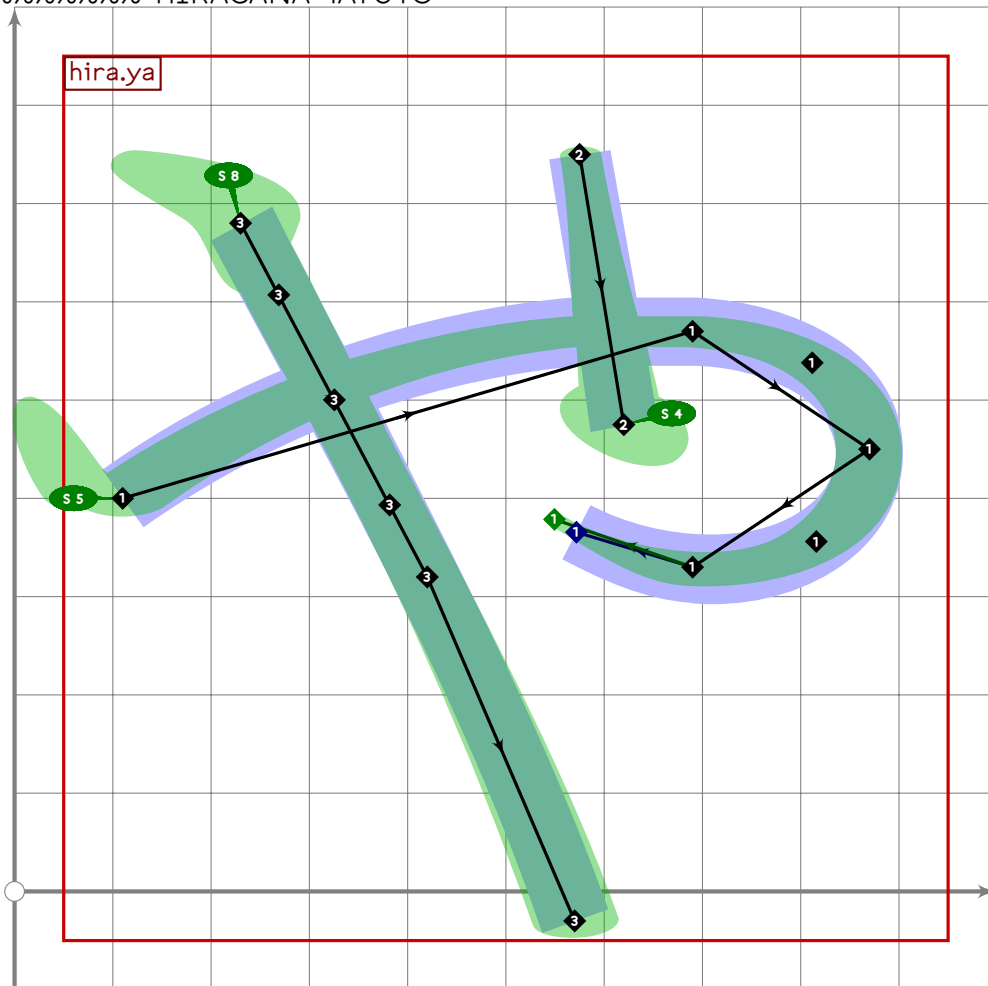
HIRA

720 enddef;

721

## Hiragana Yayuyo

722 %%%%%%%%% HIRAGANA YAYUYO



723

724 vardef hira.ya =

725 push\_pbox\_toexpand("hira.ya");

726

727 push\_stroke((110,400)..(690,570)..(870,450)..(690,330)..(520,400),  
728 (1.8,1.8)–(1.6,1.6)–(1.4,1.4)–(1.8,1.8)–(0.6,0.6));

729 set\_boserif(0,0,5);

730

731 push\_stroke((575,750)–(620,475),  
732 (1.1,1.1)–(1.6,1.6));

733 set\_boserif(0,1,4);

734

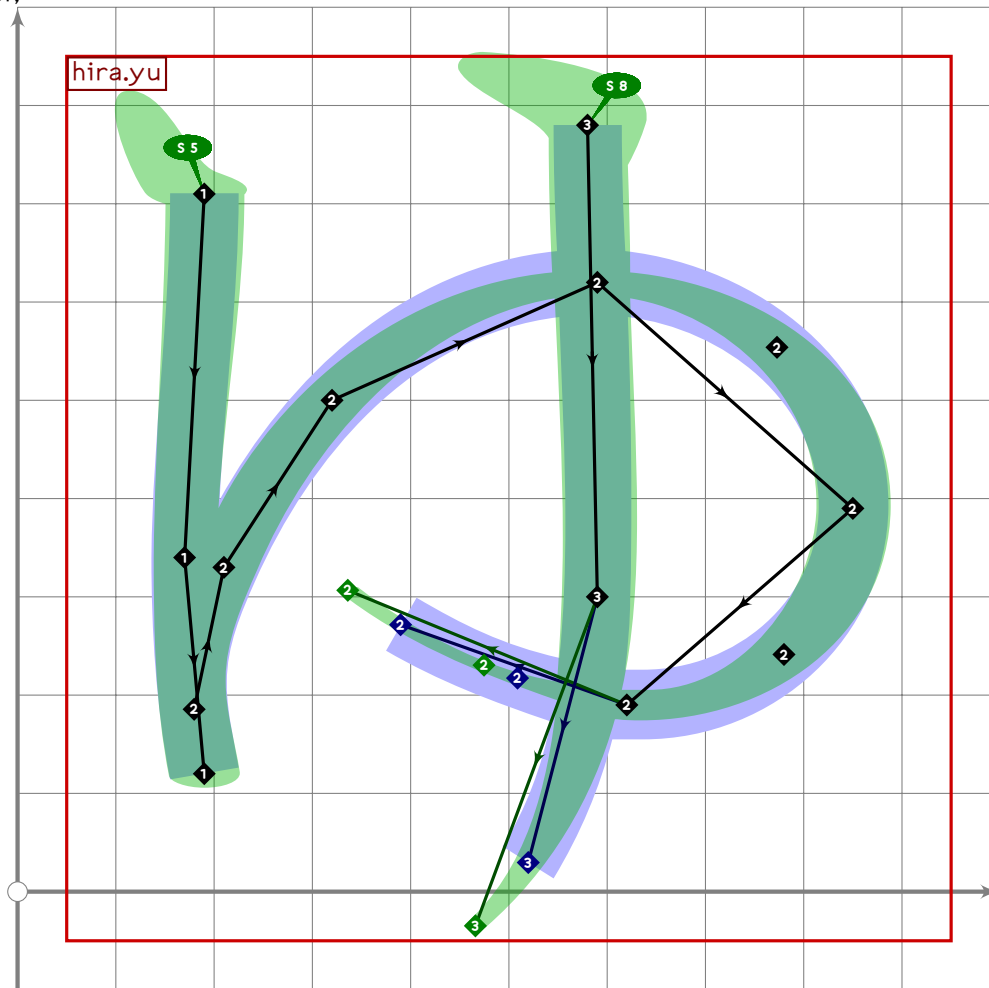
735 push\_stroke((230,680)..tension 2..(420,320)..(570,-30),  
736 (1.6,1.6)–(1.4,1.4)–(1.7,1.7));

737 set\_boserif(0,0,8);

738 expand\_pbox;

HIRA

739 enddef;



740

741 vardef hira.yu =

742 push\_pbox\_toexpand("hira.yu");

743

744 push\_stroke((190,710){down}...(170,340)..(190,120),

745 (1.6,1.6)–(1.4,1.4)–(1.5,1.5));

746 set\_boserif(0,0,5);

747

748 push\_stroke((point 1.7 of get\_stroke(0))

749 {-direction 1.7 of get\_stroke(0)}..(210,330)..

750 (320,500)..(590,620)..(850,390)..(620,190)..tension 1.3..(320,320),

751 (1.4,1.4)–(1.5,1.5)–(1.4,1.4)–(1.5,1.5)–(1.6,1.6)–

752 (1.6,1.6)–(0.77,0.77));

753

754 push\_stroke((580,780){down}...(590,300)..{dir 190}(360,-85),

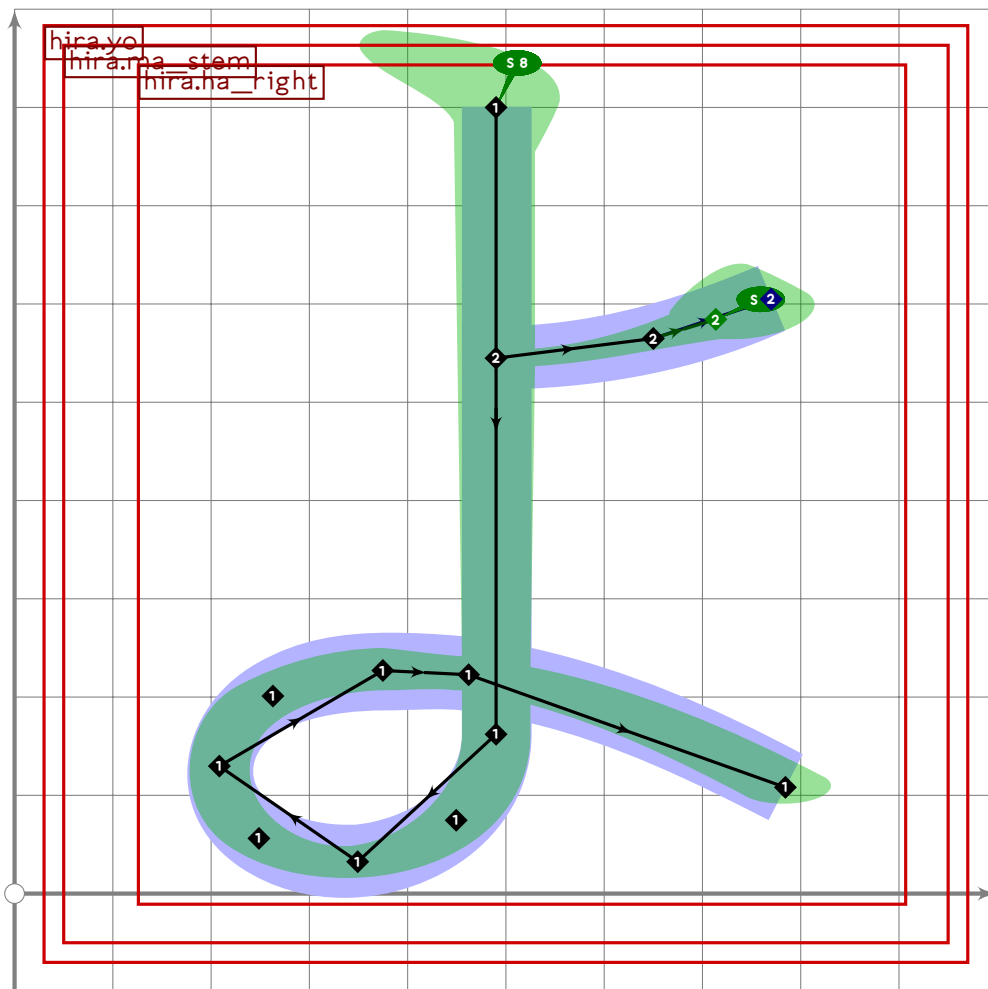
755 (1.6,1.6)–(1.5,1.5)–(0.6,0.6));

756 set\_boserif(0,0,8);

757 expand\_pbox;

758 enddef;

**HIRA**



```

759
760 vardef hira.yo =
761   push_pbox_toexpand("hira.yo");
762
763   hira.ma_stem;
764
765   replace_strokep(0)(oldp shifted (-20,0));
766   z0=point 0.4 of get_strokep(0);
767
768   push_stroke(z0..(z0+(160,20))..(z0+(280,60))+60*mincho*dir 200),
769     (1.2,1.2)-(1.4,1.4)-(1.8,1.8));
770   set_boserif(0,2,6);
771   expand_pbox;
772 enddef;
773

```

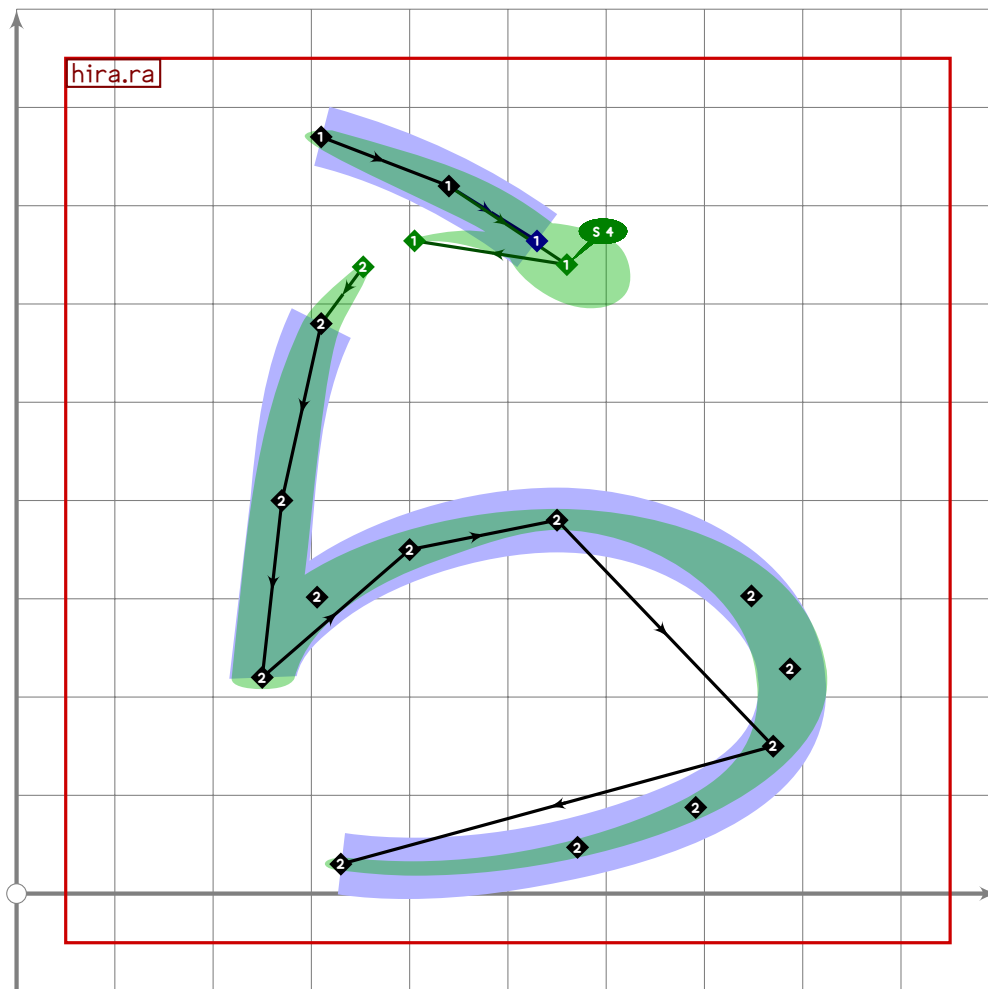
## HIRA

## Hiragana Rarirurero

```

774 %%%%%%%%% HIRAGANA RARIRURERO

```

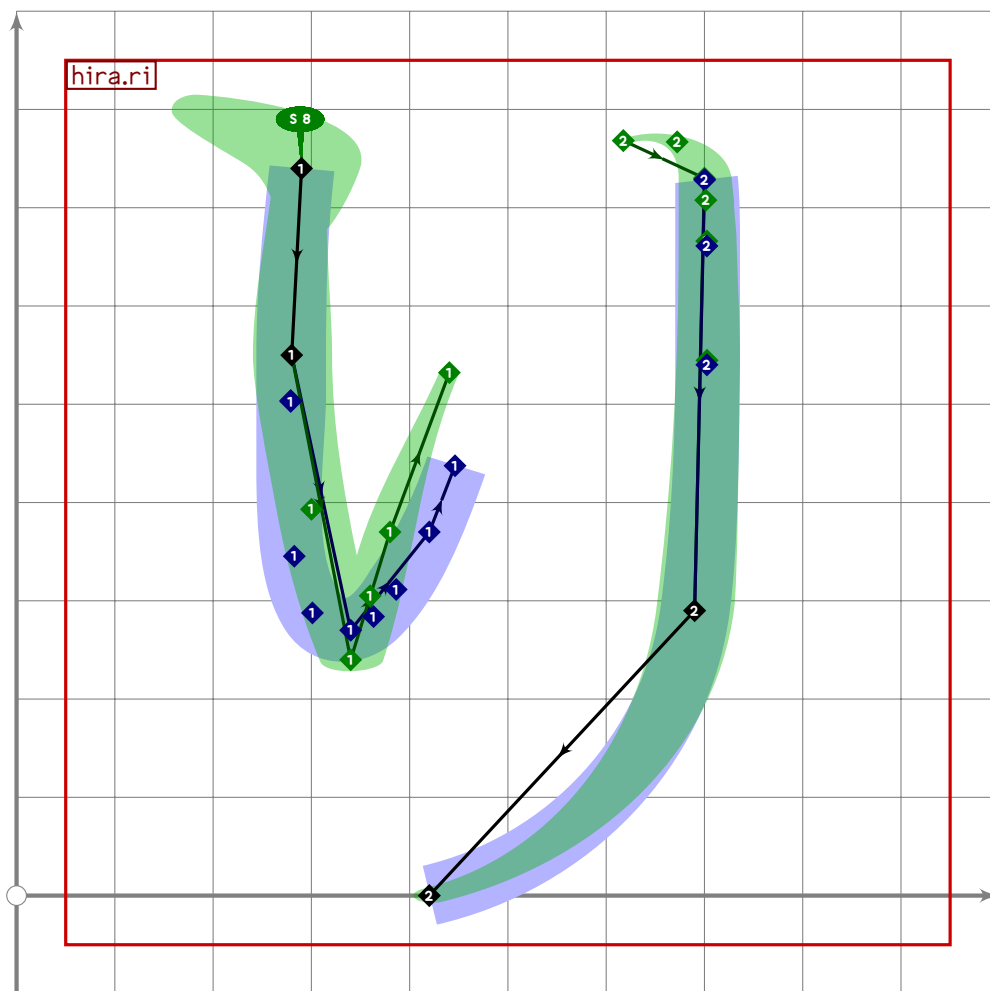


```

775
776 vardef hira.ra =
777   push_pbox_toexpand("hira.ra");
778
779   push_stroke((370,770)..(500,720)..{curl 1}{620,640}{curl 0.1}..
780     (430,650)..(370,580)..(330,400)..(310,220),
781     (1,1)-(1.6,1.6)-(2,0.78)-(0.55,0.55)-
782     (1.8,1)-(1.4,1.4)-(1.6,1.6));
783   set_boserif(0,2,4);
784
785   hira.chi_bottom;
786   set_botip(0,2,0);
787
788   replace_strokep(0)(oldp shifted (-60,0));
789   expand_pbox;
790 enddef;

```

**HIRA**



```

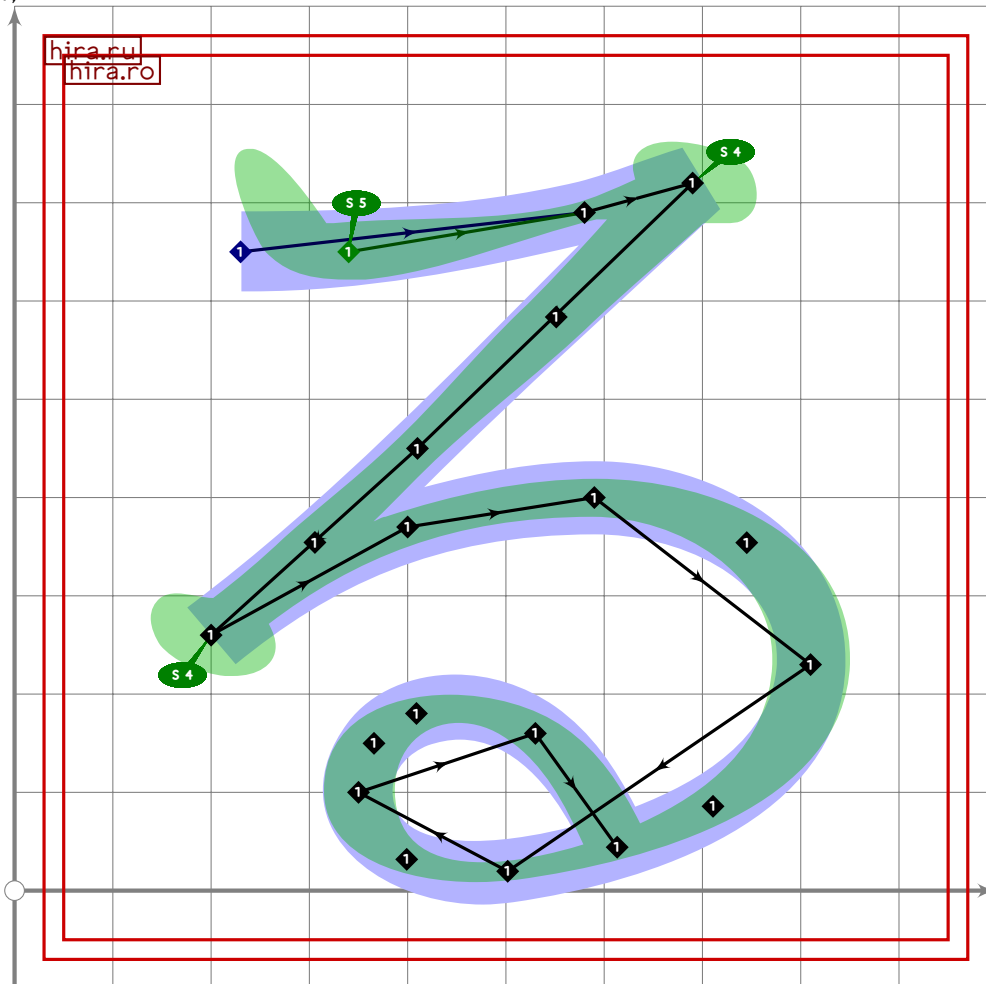
791
792 vardef hira.ri =
793   push_pbox_toexpand("hira.ri");
794
795   begingroup
796     save ripx,ripy,ripz,x,y;
797     path ripx,ripy,ripz;
798     numeric x[],y[];
799     z1=(290,740);
800     z2=(280,550);
801     z3=(340,270-30*mincho);
802     z4=(420-40*mincho,370);
803     z5=(540,710);
804     z6=(700,730);
805     ripx=z1..z2{down}..tension 1.5..z3..
806       tension 1.5..z4..z5..z6..tension 5 and 1.2..
807       (690,290)..tension 0.75 and 1.{curl 0.45}(420,0);
808     ripy=z1..z2{down}..tension 1.5..{curl 1}z3{curl 1}..
809       tension 1.5..z4..z5..z6..tension 5 and 1.2..
810       (690,290)..tension 0.75 and 1.{curl 0.45}(420,0);
811     push_stroke(interpath(mincho,ripx,ripy),

```

```

812      (1.3,1.3)–(1.6,1.6)–(1.4,1.4)–(1.2,1.2)–(0.4,0.2)–
813      (1.5,0.99)–(1.6,1.6)–(1,1));
814      set_boserif(0,0,8);
815      endgroup;
816      expand_pbox;
817  enddef;

```

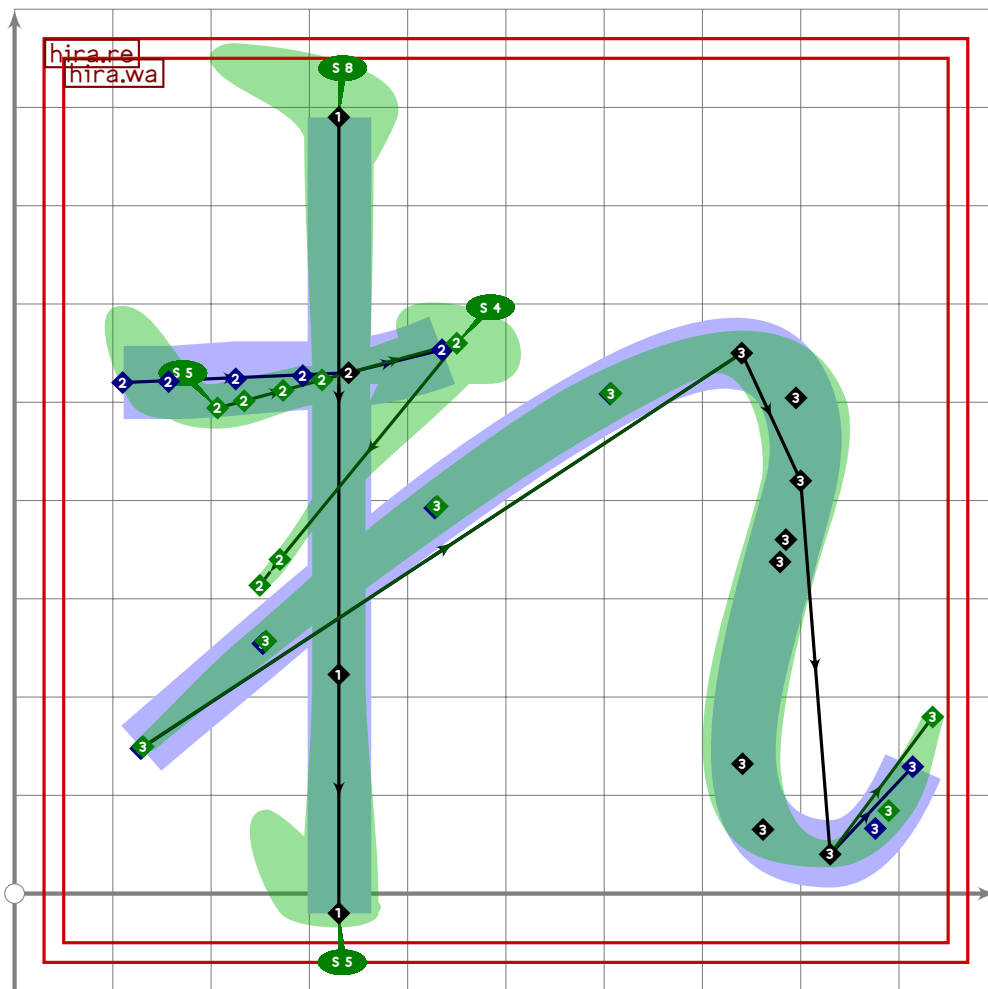


```

818
819  vardef hira.ru =
820    push_pbox_toexpand("hira.ru");
821
822    hira.ro;
823
824    replace_stroke(0)((subpath (0,7.8) of oldp)..(350,100)..(530,160)..
825      {curl 0.2}(point 7.6 of oldp));
826    replace_stroke(0)((2.6,2.6)–(1.2,1.2)–(1.9,1.9)–
827      (1.3,1.3)–(1.6,1.6)–
828      (1.5,1.5)–(1.9,1.9)–(1.6,1.6)–
829      (1.2,1.2)–(1.5,1.5)–(1.4,1.4));
830    expand_pbox;
831  enddef;

```

**HIRA**

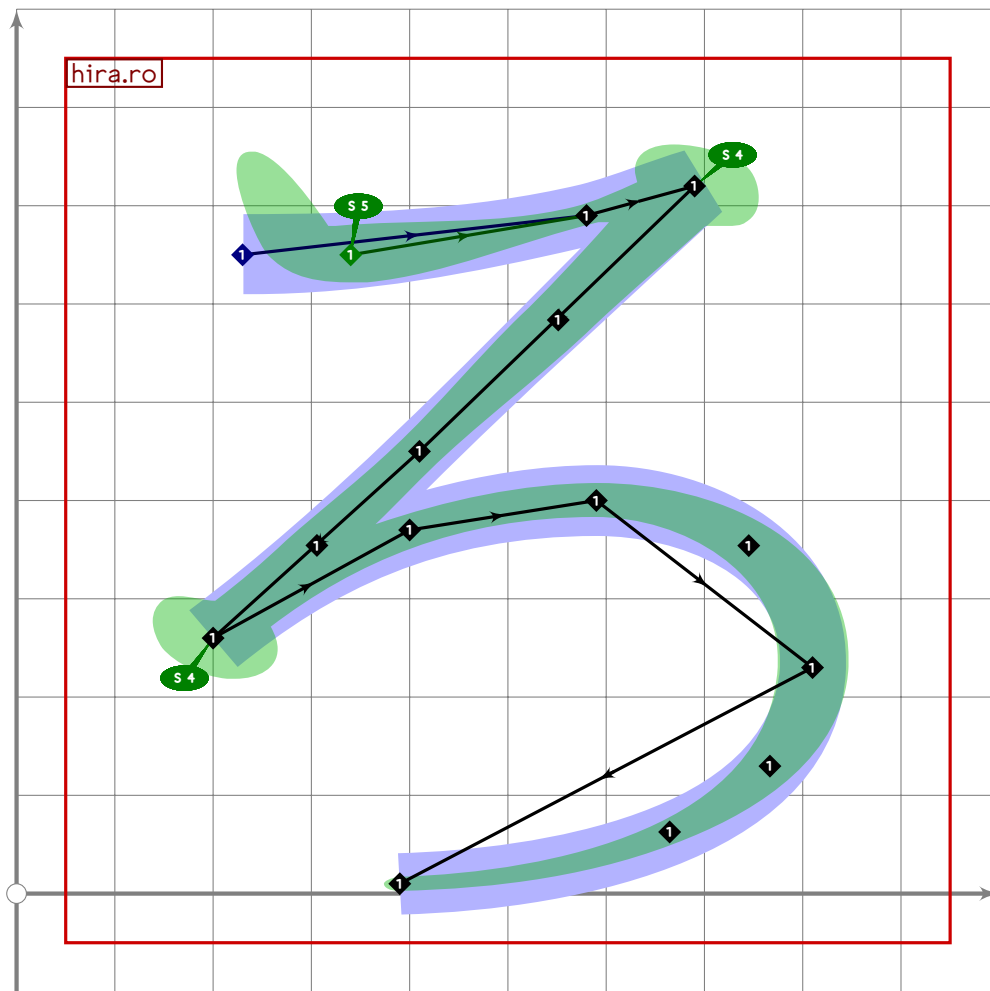


```

832
833 vardef hira.re =
834   push_pbox_toexpand("hira.re");
835
836   hira.wa;
837
838   replace_strokep(0)((subpath (0,4) of oldp){curl 0}..
839     tension 2..(740,550)..(800,420)..
840     (830,40){right}..tension 1.5..{curl 0}(960,270));
841   replace_strokeq(0)((2,2)-(1.6,1.6)-(2.7,0.9)-
842     (0.84,0.7)-(0.79,0.97)-
843     (2.1,2.1)-(1.6,1.6)-(1.5,1.5)-(0.5,0.5));
844   set_boserif(0,2,4);
845   expand_pbox;
846 enddef;

```

**HIRA**



```

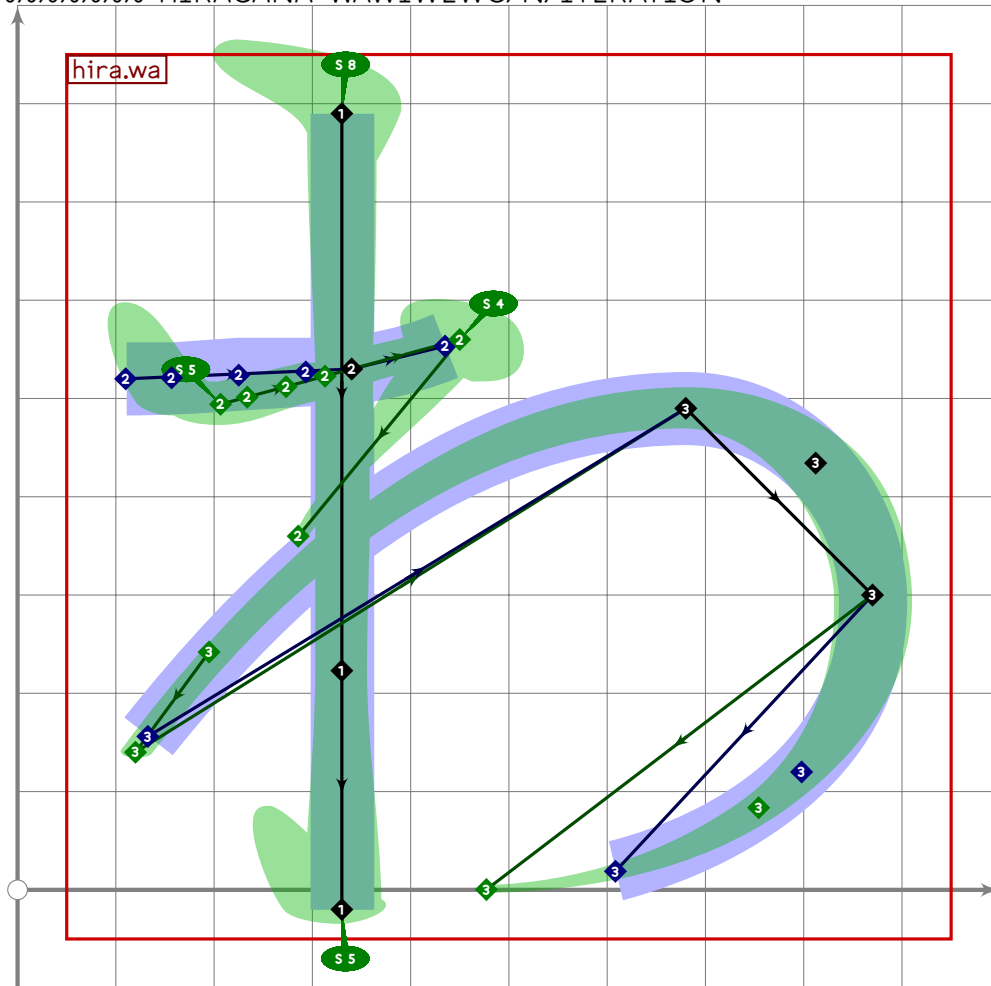
847
848 vardef hira.ro =
849   push_pbox_toexpand("hira.ro");
850
851   push_stroke((230+110*mincho,650)..(580,690)..{curl 1}{690,720}{curl 1}..
852     (410,450)..{curl 1}{200,260}{curl 1}..
853     (400,370)..(590,400){right}{(810,230)..
854     tension 1.1..{curl 0}{390,10},
855     (2.6,2.6)-(1.2,1.2)-(1.9,1.9)-
856     (1.3,1.3)-(1.6,1.6)-
857     (1.5,1.5)-(1.7,1.7)-(1.5,1.5)-(1,1));
858   set_botip(0,2,0);
859   set_botip(0,4,0);
860   set_boserif(0,0,5);
861   set_boserif(0,2,4);
862   set_boserif(0,4,4);
863   expand_pbox;
864 enddef;
865

```

**HIRA**

## Hiragana Wawiwewo/N/Iteration

866 %%%%%%%%% HIRAGANA WAWIWEWO/N/ITERATION



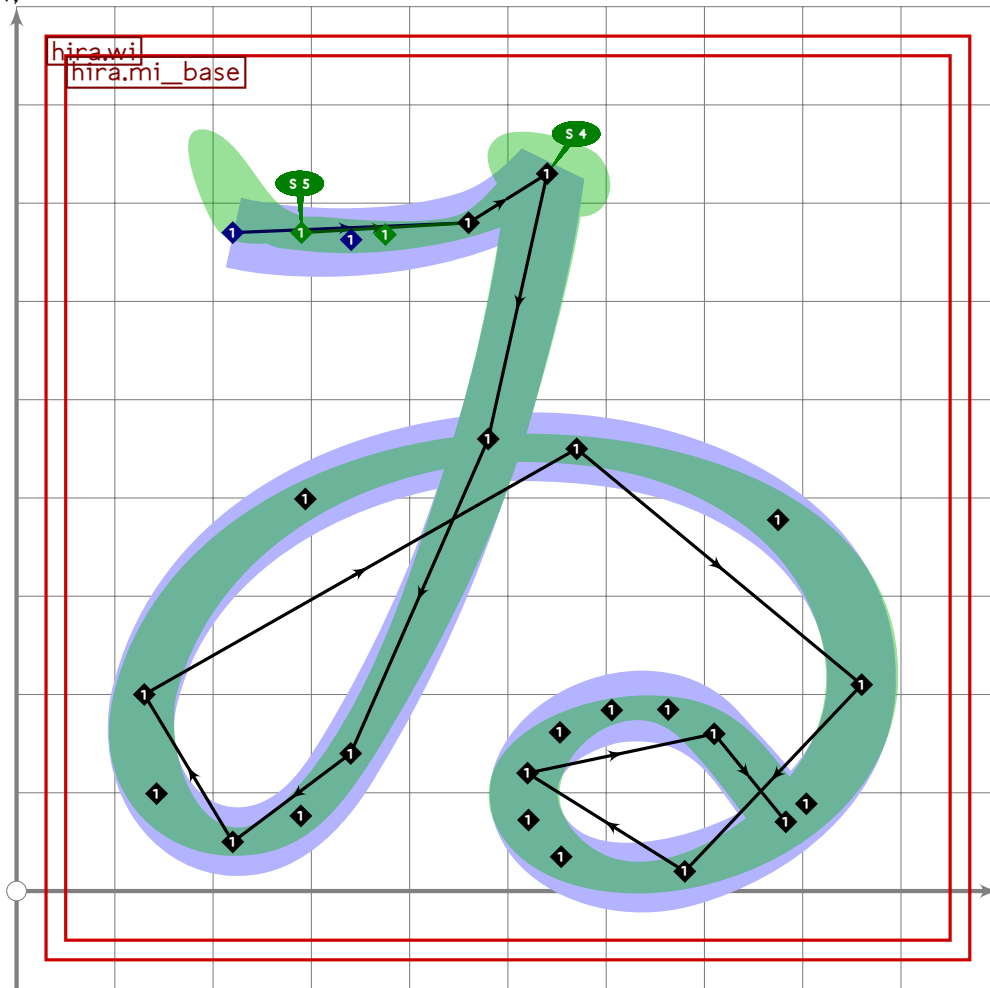
```

867
868 vardef hira.wa =
869   push_pbox_toexpand("hira.wa");
870
871   push_stroke(((330,790)-(0.7[(330,790),(330,-20)])-(330,-20),
872     (1.5,1.5)-(1.2,1.2)-(1.6,1.6));
873   set_boserif(0,0,8);
874   set_boserif(0,2,5);
875
876   push_stroke(((110,520)+100*mincho*dir -15)..tension 2..(340,530)..
877     {curl 1}(450,560){curl 1}..
878     (270,340)..{curl 1}(120,140){curl 0.2}..
879     (680,490){right}..(870,300)..{curl 0.2}(450,0),
880     (2,2)-(1.6,1.6)-(2.2,0.9)-
881     (0.7,0.7)-(0.97,0.97)-
882     (2,2)-(1.6,1.6)-(0.8,0.8));
883   set_botip(0,2,0);
884   set_botip(0,4,0);
885   set_boserif(0,0,5);

```

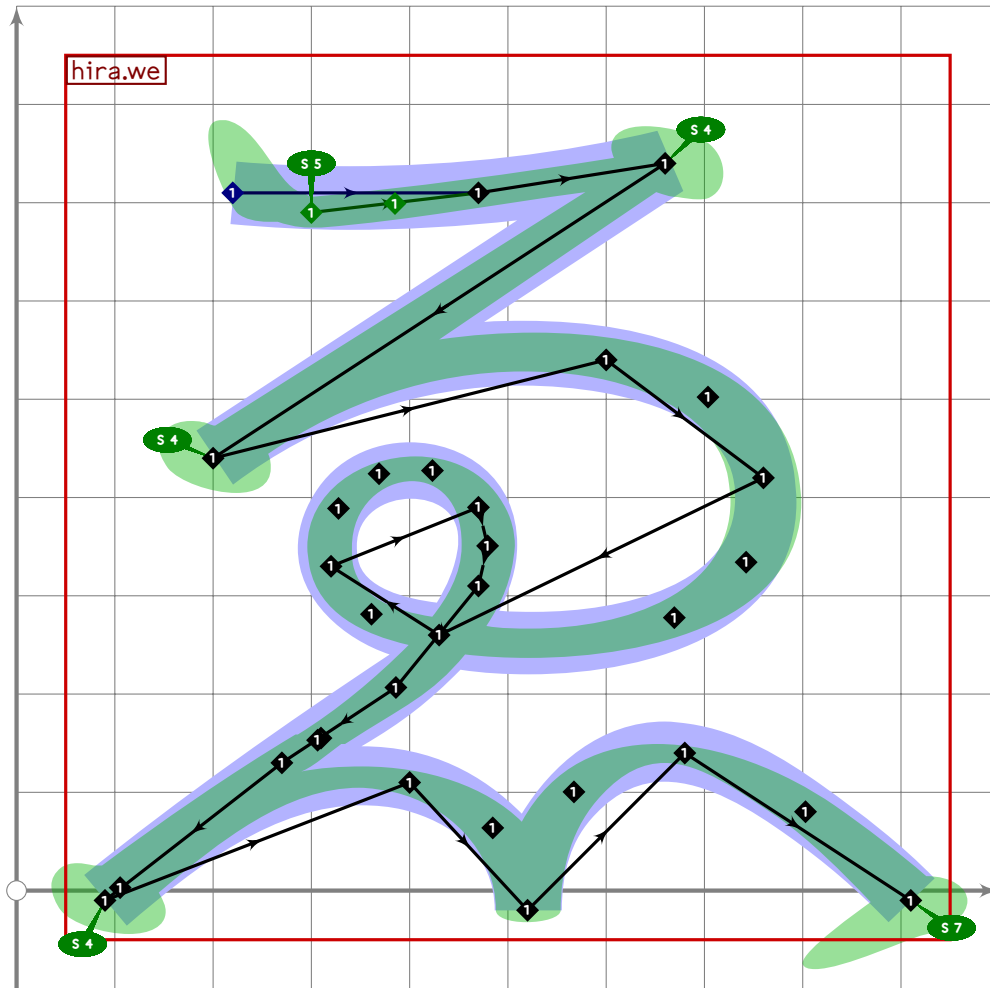
HIRA

```
886 set_boserif(0,2,4);
887 expand_pbox;
888 enddef;
```



```
889
890 vardef hira.wi =
891   push_pbox_toexpand("hira.wi");
892
893   hira.mi_base;
894
895   replace_strokep(0)((subpath (0,6) of oldp)..(570,450)..(860,210)..(680,20)..
896     (520,120)..tension 1.2..(710,160));
897   replace_strokep(0)(oldp..{curl 0.3}(point 8.6 of oldp));
898
899   replace_strokeq(0)((1.7,1.7)-(1.3,1.3)-(1.6,1.6)-
900     (1.5,1.5)-(1.2,1.2)-(1.5,1.5)-(1.4,1.4)-
901     (1.6,1.6)-(1.5,1.5)-(1.7,1.7)-(1.4,1.4)-(1.5,1.5));
902   expand_pbox;
903 enddef;
```

**HIRA**



```

904
905 vardef hira.we =
906   push_pbox_toexpand("hira.we");
907
908   push_stroke((220+80*mincho,710-20*mincho)..(470,710)..{curl 1}(660,740)-
909     (200,440){curl 1}..
910     (600,540)..(760,420)..(430,260)..(320,330)..(470,390)..(470,310)..
911     (270,130)..{curl 0}(90,-10){curl 0.1}..
912     (400,110)..{down}(520,-20){up}..
913     (680,140)..tension 1.3..{curl 0.2}(910,-10),
914     (1.8,1.8)-(1.6,1.6)-(1.5,1.5)-(1.9,1.9)-
915     (2,2)-(1.6,1.6)-(1.7,1.7)-(1.2,1.2)-(1.3,1.3)-(1.35,1.35)-
916     (1.4,1.4)-(1.5,1.5)-(1.3,1.3)-(1.8,1.8)-
917     (1.4,1.4)-(1.5,1.5)-
918     (1.3,1.3)-(1.7,1.7));
919   replace_strokep(0)(insert_nodes(olddp)(8.5,9.5));
920   set_bosize(0,90);
921   set_botip(0,2,0);
922   set_botip(0,3,0);
923   set_botip(0,13,0);
924   set_botip(0,15,0);

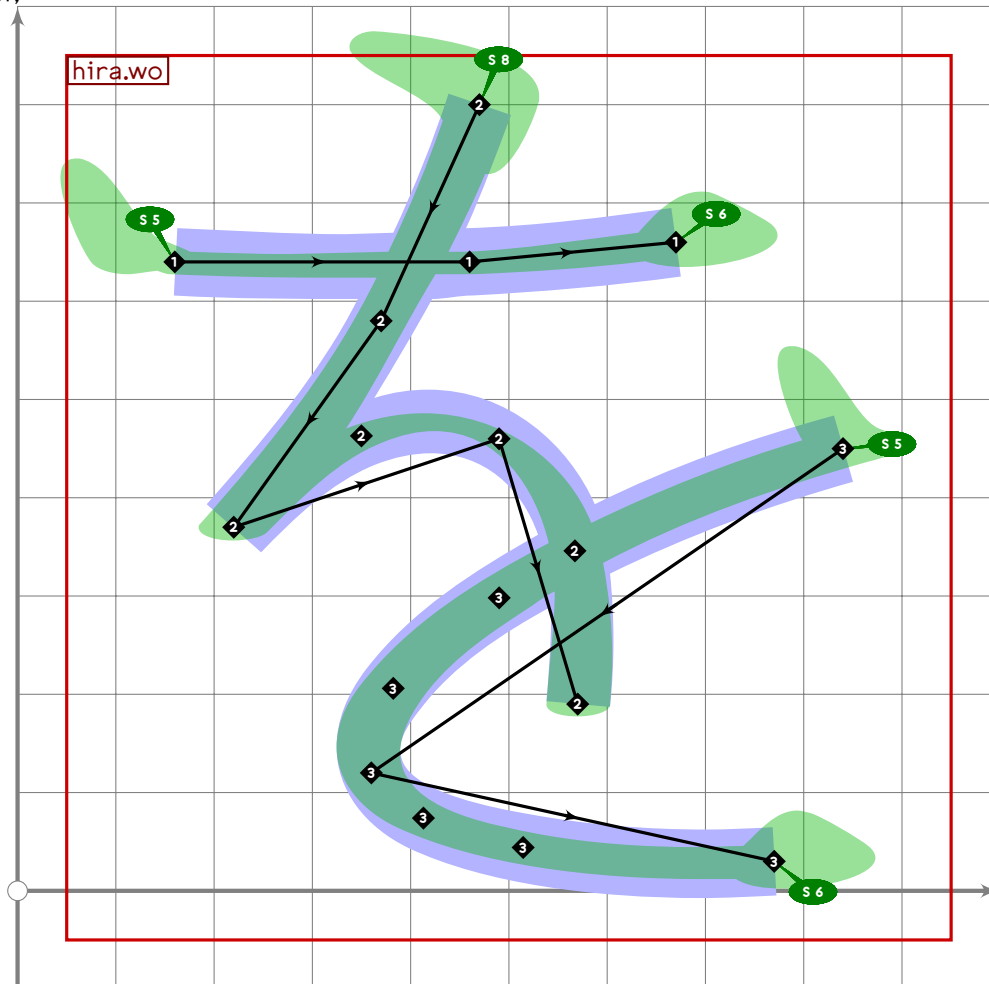
```

**HIRA**

```

925 set_boserif(0,0,5);
926 set_boserif(0,2,4);
927 set_boserif(0,3,4);
928 set_boserif(0,13,4);
929 set_boserif(0,17,7);
930 expand_pbox;
931 enddef;

```



```

932
933 vardef hira.wo =
934   push_pbox_toexpand("hira.wo");
935
936   push_stroke((160,640)..(460,640)..(670,660),
937     (1.3,1.3)–(1.4,1.4)–(1.6,1.6));
938   set_boserif(0,0,5);
939   set_boserif(0,2,6);
940
941   push_stroke((470,800)..(370,580)..{curl 1}(220,370){curl 0}..
942     (490,460)..{curl 0}(570,190),
943     (1.4,1.4)–(1.3,1.3)–(1.5,1.5)–(1.01,1.01)–(1.4,1.4));
944   set_botip(0,2,0);
945   set_boserif(0,0,8);

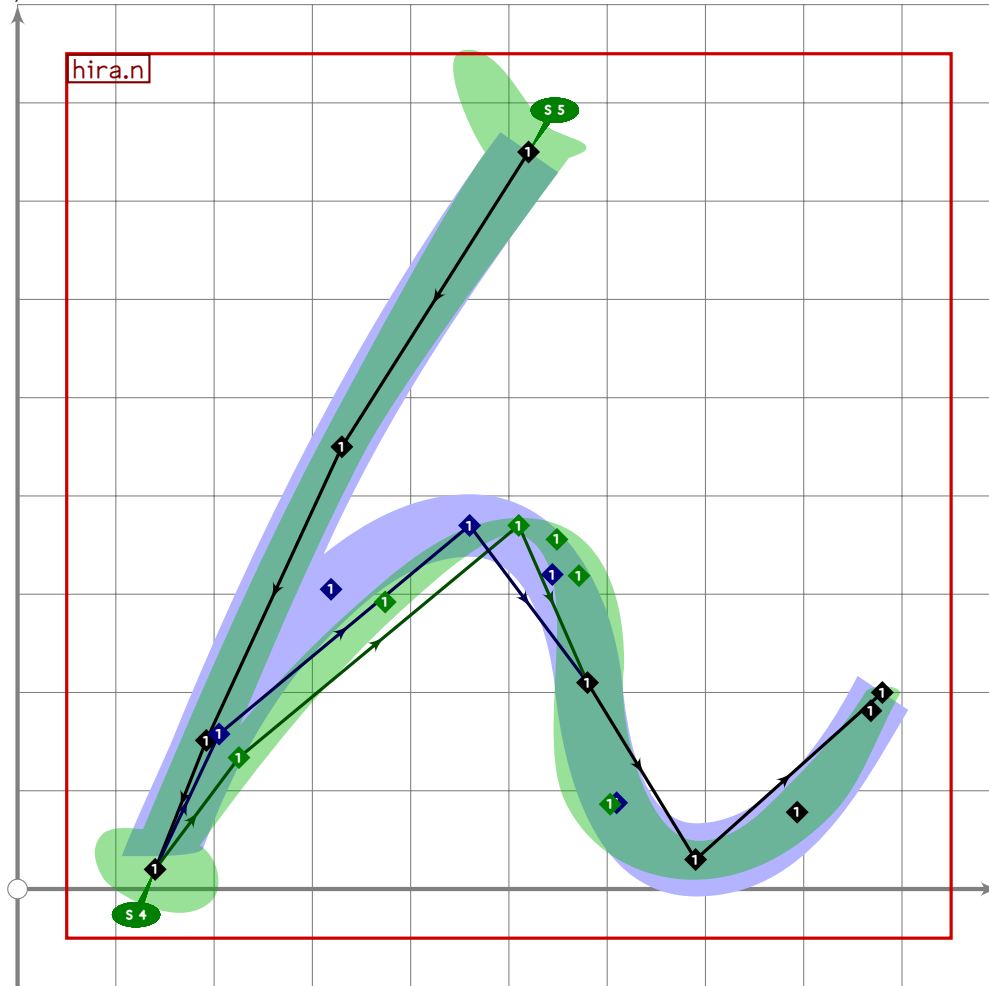
```

**HIRA**

```

946
947  push_stroke((840,450){curl 0.017}..tension 1 and 2..(360,120)..
948    tension 2 and 1.{curl 0.03}(770,30),
949    (1,7,1,7)-(1,4,1,4)-(1,7,1,7));
950  set_boserif(0,0,5);
951  set_boserif(0,2,6);
952  expand_pbox;
953 enddef;

```



## HIRA

```

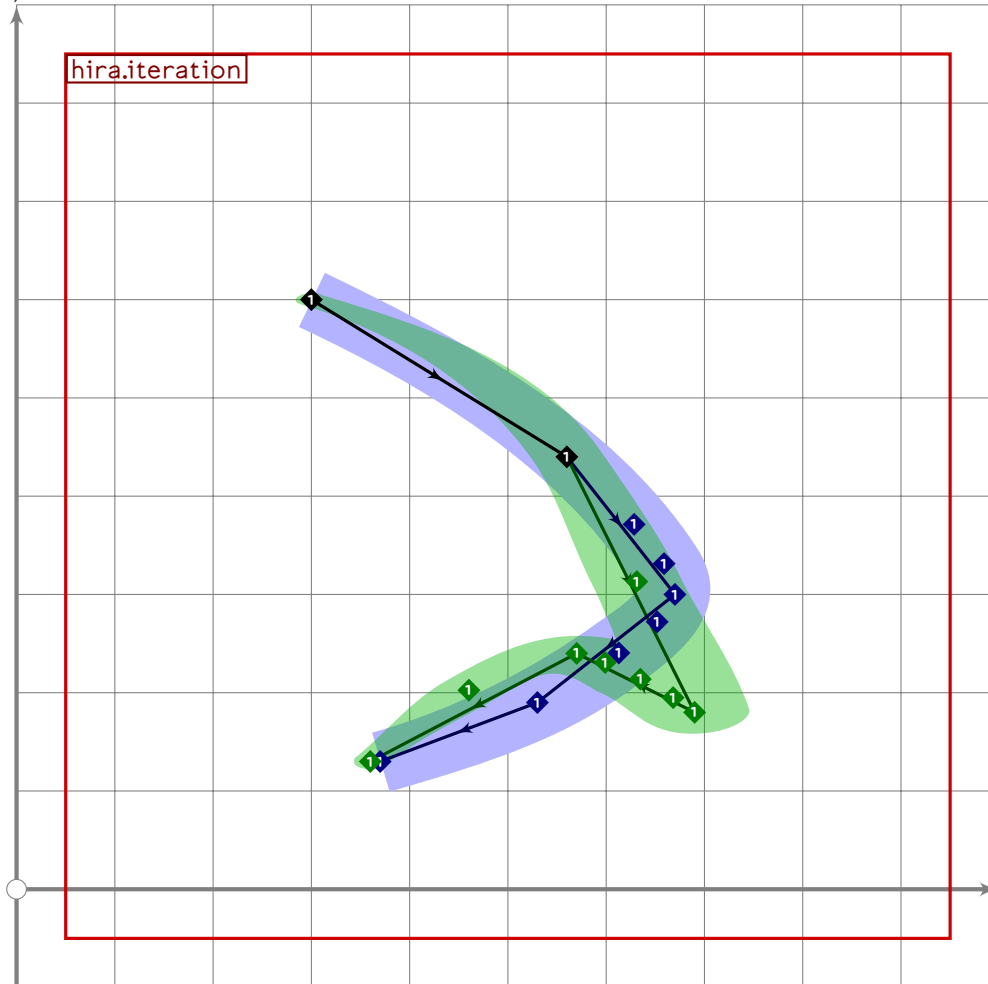
954
955 vardef hira.n =
956   push_pbox_toexpand("hira.n");
957
958   push_stroke((520,750)..(330,450){curl 0.2}(140,20){curl 0.1}..
959     tension (1.2+0.6*mincho)..(460+50*mincho,370){right}..
960     (580,210)..(690,30){right},
961     (1,7,1,7)-(1,2,1,2)-(1,3,1,3)-
962     (1,1,1,1)-(1,5,1,5)-(1,9,1,9)-(1,1));
963   replace_strokep(0)(oldp{right}..(880,200){direction 0.5 of oldp});
964   replace_strokep(0)(insert_nodes(oldp)(1,7,2,3));
965   replace_strokeq(0)(insert_nodes(oldq)(1,7,2,3));
966   set_botip(0,3,0);

```

```

967 set_boserif(0,0,5);
968 set_boserif(0,3,4);
969 expand_pbox;
970 enddef;

```



```

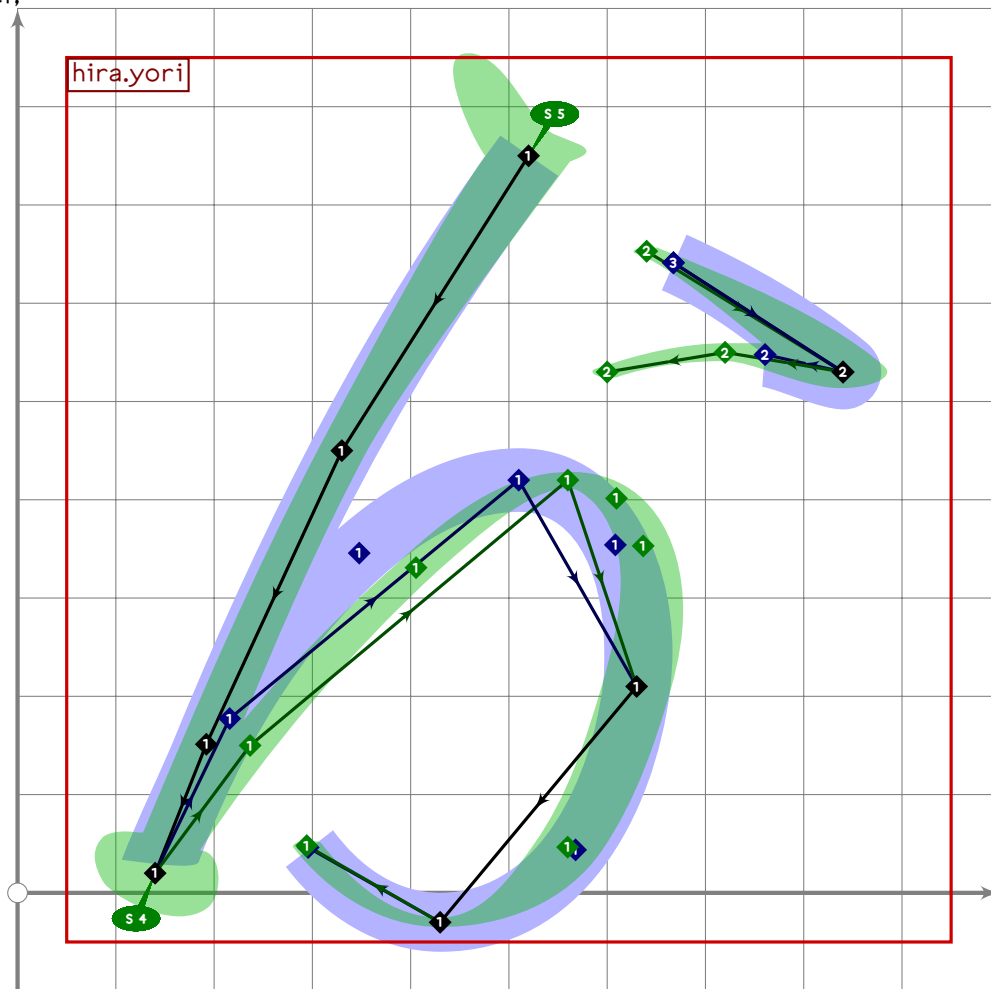
971
972 vardef hira.iteration =
973   push_pbox_toexpand("hira.iteration");
974
975   push_stroke(begingroup
976     save ripx,ripy;
977     path ripx,ripy;
978     ripx:=(300,600){curl 0.2}..(560,440)..
979       tension 1.5 and 2..(670,300)..
980       tension 2 and 1.5..(530,190)..{curl 0.2}(370,130);
981     ripy:=(300,600){curl 0.2}..(560,440)..
982       tension 1.5..{curl 1}(690,180){curl 1}..
983       tension 2 and 1.5..(570,240)..tension 14..{curl 0}(360,130);
984     interpath(mincho,ripx,ripy)
985   endgroup,
986   (1,1)-(1.5,1.5)-(2,2)-(1.9,1.9)-(1,1));
987   set_botip(0,2,0);

```

**HIRA**

U+309F  
tsuku.uni309F

```
988 expand_pbox;
989 enddef;
```



```
990
991 vardef hira.yori =
992   push_pbox_toexpand("hira.yori");
993
994   push_stroke((520,750)..(330,450){curl 0.2}(140,20){curl 0.1}..
995     tension (1.2+0.6*mincho)..(510+50*mincho,420){right}..
996     (630,210)..(430,-30){left}..tension 1.3..(290,600)..
997     {curl 0.1}(840,530){curl 1}..(720,550)..(600,530),
998     (1.7,1.7)-(1.2,1.2)-(1.3,1.3)-
999     (1.1,1.1)-
1000     (1.5,1.5)-(1.5,1.5)-(-1,-0.4)-
1001     (1.8,1.6)-(1.6,0.7)-(1.9,0));
1002   replace_strokep(0)(insert_nodes(oldp)(1.7,2.3));
1003   replace_strokeq(0)(insert_nodes(oldq)(1.7,2.3));
1004   set_botip(0,3,0);
1005   set_botip(0,9,0);
1006   set_boserif(0,0,5);
1007   set_boserif(0,3,4);
1008   expand_pbox;
```

**HIRA**

1009 enddef;

## iching.mp

```
1 %
2 % I Ching characters for Tsukurimashou
3 % Copyright (C) 2011 Matthew Skala
4 %
5-29 [Standard copyright notice]
30
31 inclusion_lock(iching);
32
33 

---


34
35 iching.size:=680;
36
37 vardef make_iching_xform(expr numlines) =
38   transform iching_xform;
39   numeric x[];
40
41   x2-x1=iching.size;
42   (x1+x2)/2=500;
43
44   (-1,(numlines+1)/2) transformed iching_xform=
45     (x1,0.5[latin_wide_low_h,latin_wide_high_h]);
46   (1,(numlines+1)/2) transformed iching_xform=
47     (x2,0.5[latin_wide_low_h,latin_wide_high_h]);
48   (-1,(numlines+1)/2-2.5) transformed iching_xform=
49     (x1,latin_wide_low_h);
50 enddef;
51
52 vardef iching.line(expr line,numlines,linetype) =
53   make_iching_xform(numlines);
54   if linetype=0:
55     push_stroke(((1,line)-(-0.28,line)) transformed iching_xform,
56       (2,2)-(2,2));
57     push_stroke(get_strokep(0) reflectedabout (centre_pt,centre_pt+down),
58       (2,2)-(2,2));
59   else:
60     push_stroke(((1,line)-(1,line)) transformed iching_xform,
61       (2,2)-(2,2));
62   fi;
63   push_anchor(anc_iching_line(line),
64     identity shifted (((1,line) transformed iching_xform)
65       +(1000-iching.size)*0.25*right));
66 enddef;
67
68 % WARNING, nonstandard calling convention, simply returns a path to fill
69 vardef iching.dot(expr line,numlines) =
70   begingroup;
```

```

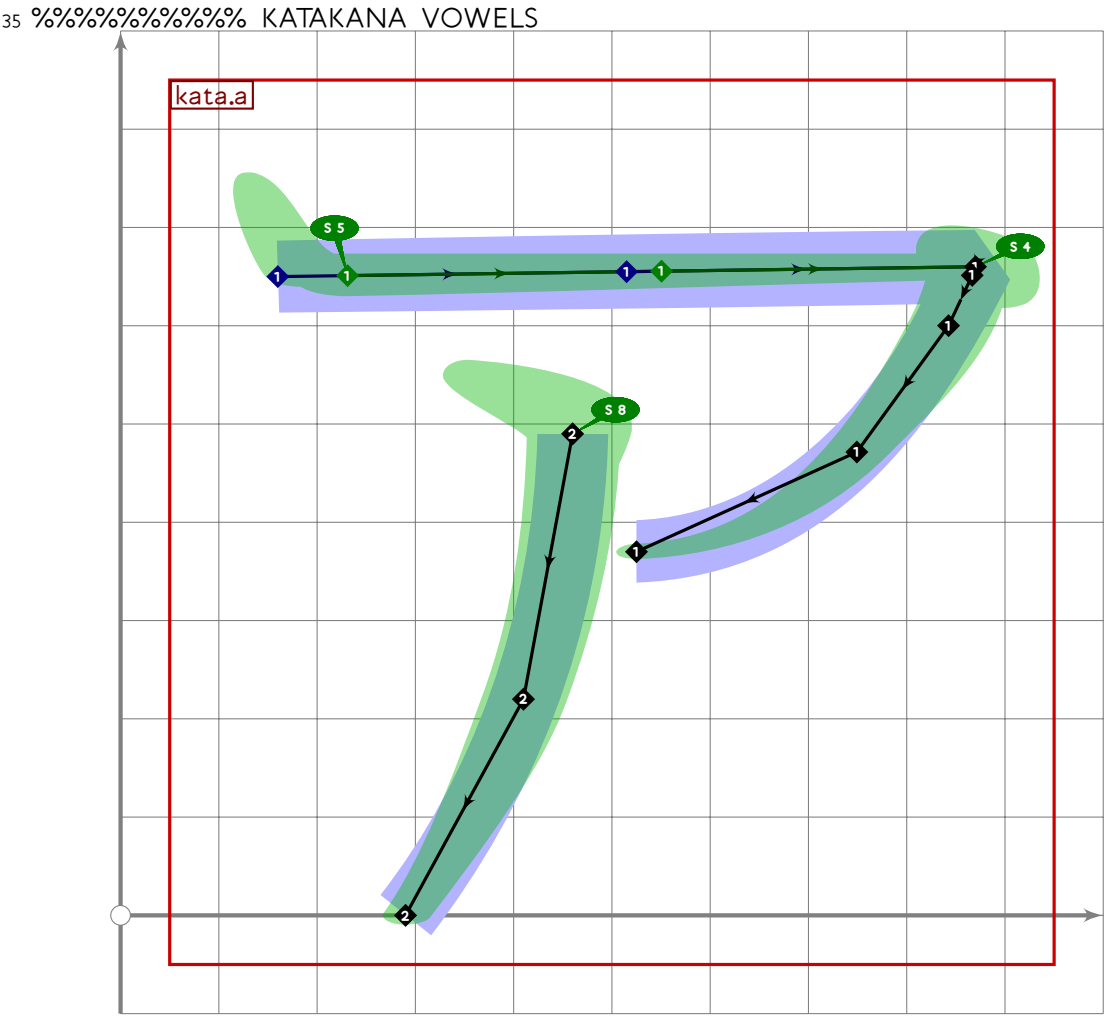
71     make_iching_xform(numlines);
72     (fullcircle scaled (tsu_punct_size*1.10)
73       shifted (((1,line) transformed iching_xform)
74               +(1000-iching.size)*0.25*right))
75   endgroup
76 enddef;

```

katakana.mp

```
1 %
2 % Katakana for Tsukurimashou
3 % Copyright (C) 2011, 2012 Matthew Skala
4 %
5-29 [Standard copyright notice]
30
31 inclusion_lock(katakana);
32
33
34
```

Katakana Vowels

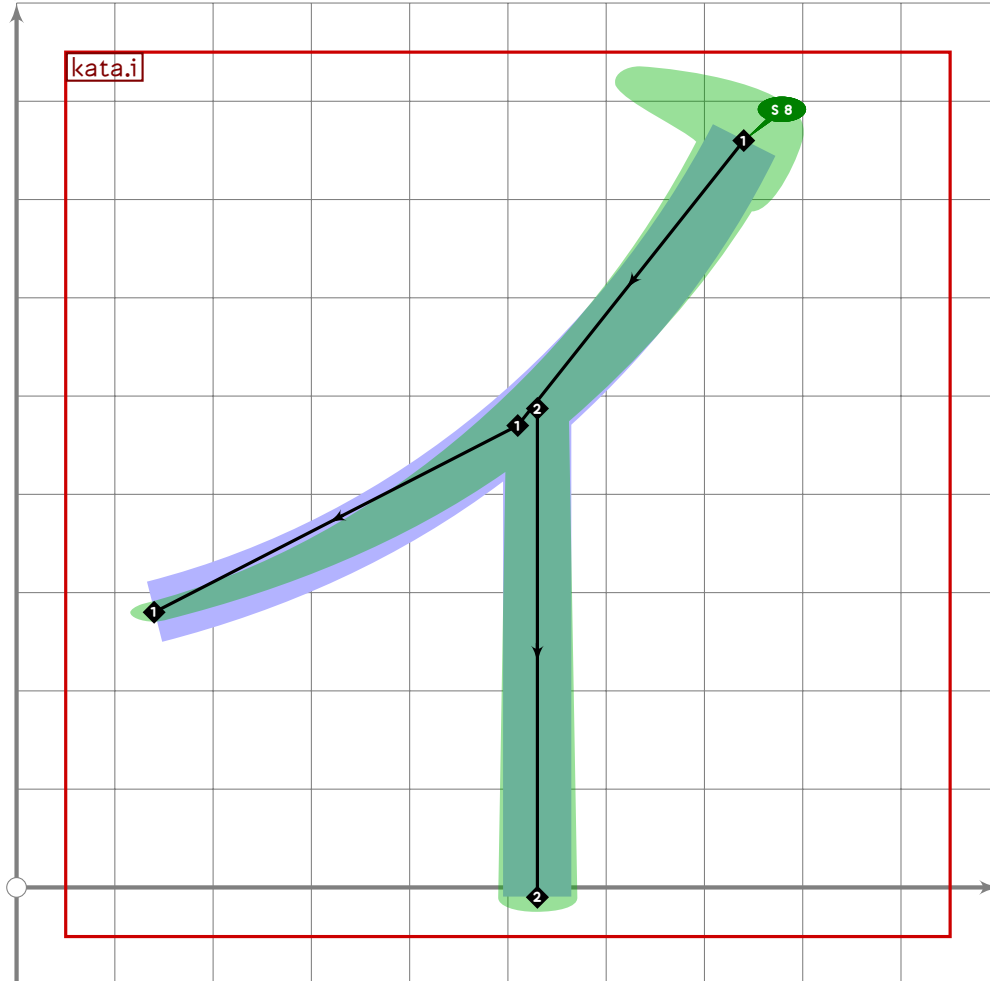


```
36
37 vardef kata.a =
38   push_pbox_toexpand("kata.a");
39
40   kata.fu_stroke((160,650),(870,660),(525,370));
41
```

```

42  push_stroke((460,490)..(410,220)..(290,0),
43    (1.8,1.8)–(1.7,1.7)–(1.2,1.2));
44  set_boserif(0,0,8);
45  expand_pbox;
46  enddef;

```

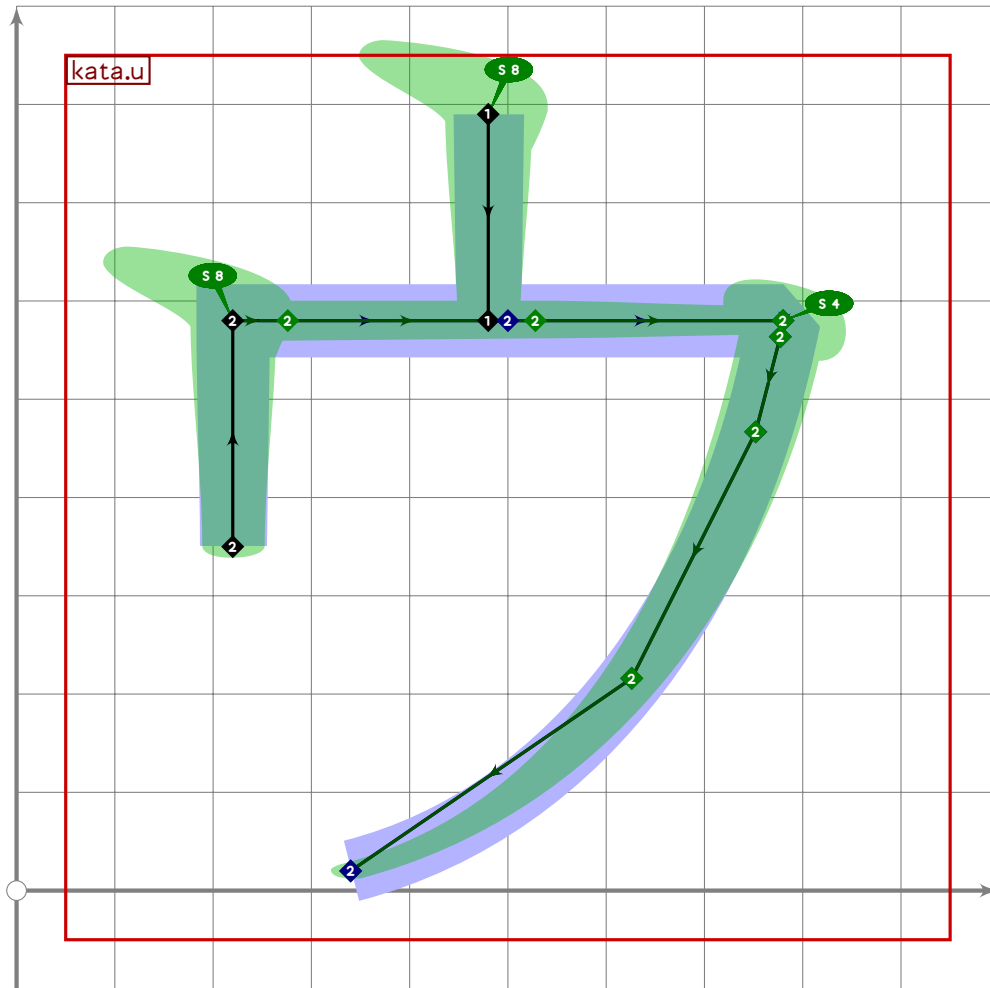


```

47
48  vardef kata.i =
49    push_pbox_toexpand("kata.i");
50
51    push_stroke((740,760)..(510,470)..(140,280),
52      (1.8,1.8)–(1.7,1.7)–(1.2,1.2));
53    set_boserif(0,0,8);
54
55    push_stroke((get_strokep(0) intersectionpoint
56      ((530,infinity)–(530,infinity)))–(530,10),
57      (1.4,1.4)–(1.6,1.6));
58    expand_pbox;
59  enddef;

```

**KATA**



```

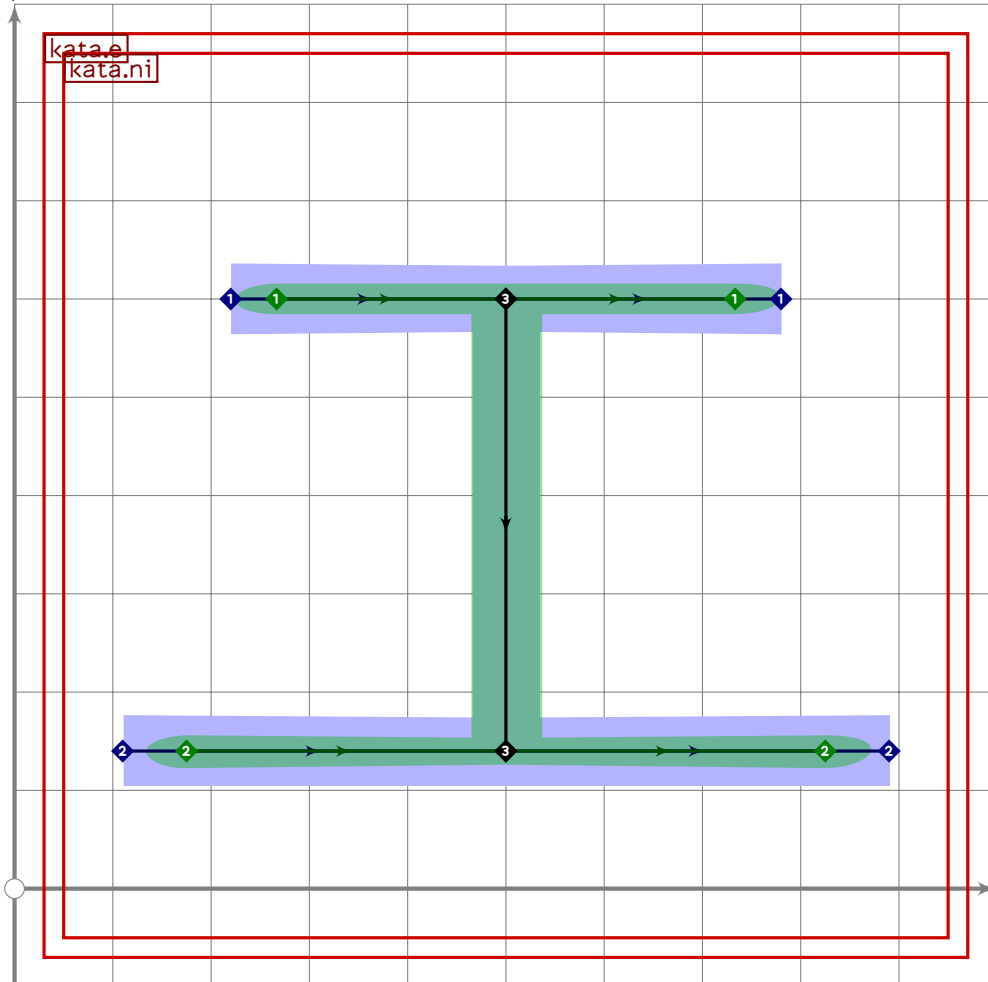
60
61 vardef kata.u =
62   push_pbox_toexpand("kata.u");
63
64   push_stroke((480,790)-(480,580),
65     (1,7,1,7)-(1,4,1,4));
66   set_boserif(0,0,8);
67
68   kata.fu_stroke((220,580),(780,580),(340,20));
69   if mincho>0.01:
70     replace_strokep(0)((220,350)-(220,580)-oldp);
71     replace_strokeq(0)((1,4,1,4)-(1,7,1,7)-oldq);
72     set_botip(0,2,1);
73     set_botip(0,4,0);
74     set_boserif(0,2,whatever);
75     set_boserif(0,4,4);
76   else:
77     replace_strokep(0)((220,350)-oldp);
78     replace_strokeq(0)((1,4,1,4)-oldq);
79     set_botip(0,1,1);
80     set_botip(0,3,0);

```

```

81   set_boserif(0,3,4);
82   fi;
83   set_boserif(0,0,whatever);
84   set_boserif(0,1,8);
85   expand_pbox;
86 enddef;

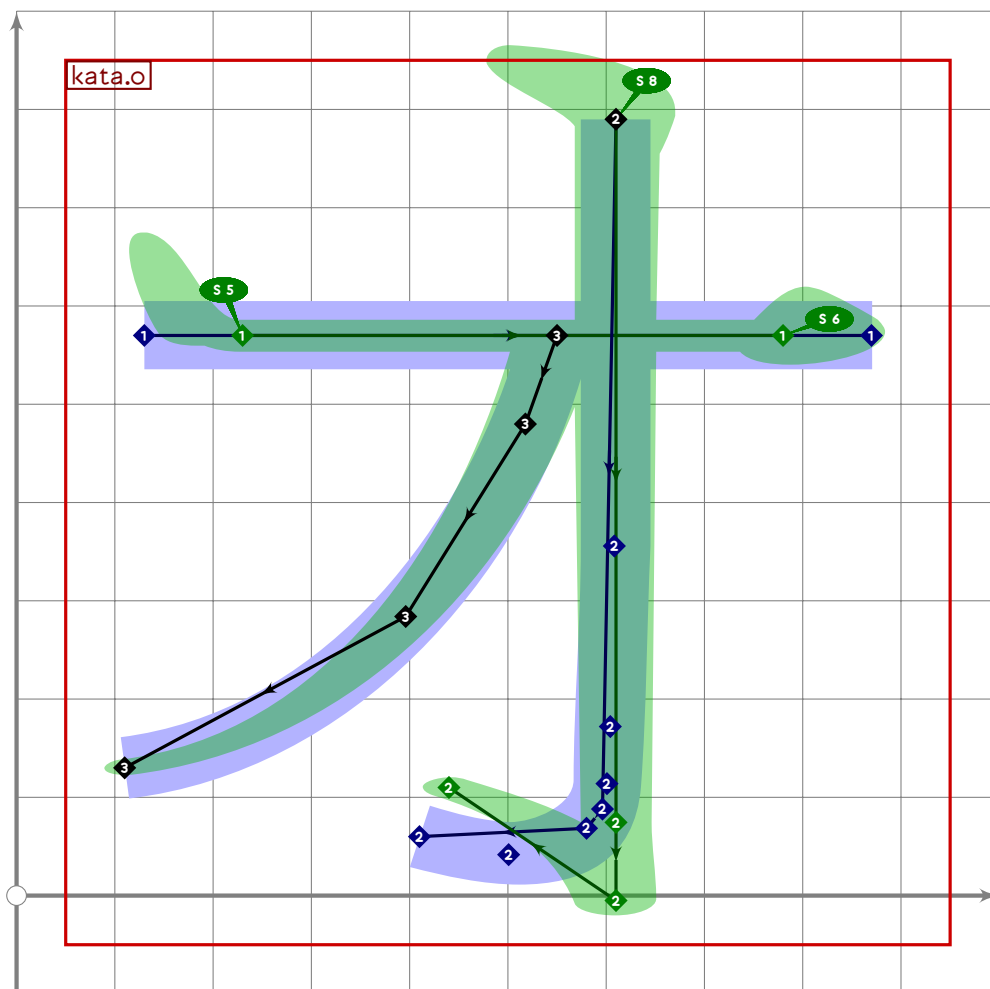
```



```

87
88 vardef kata.e =
89   push_pbox_toexpand("kata.e");
90
91   kata.ni;
92
93   push_stroke((point 1 of get_stroke(-1))-(point 1 of get_stroke(0)),
94     (1.5,1.5)-(1.5,1.5));
95   expand_pbox;
96 enddef;

```



```

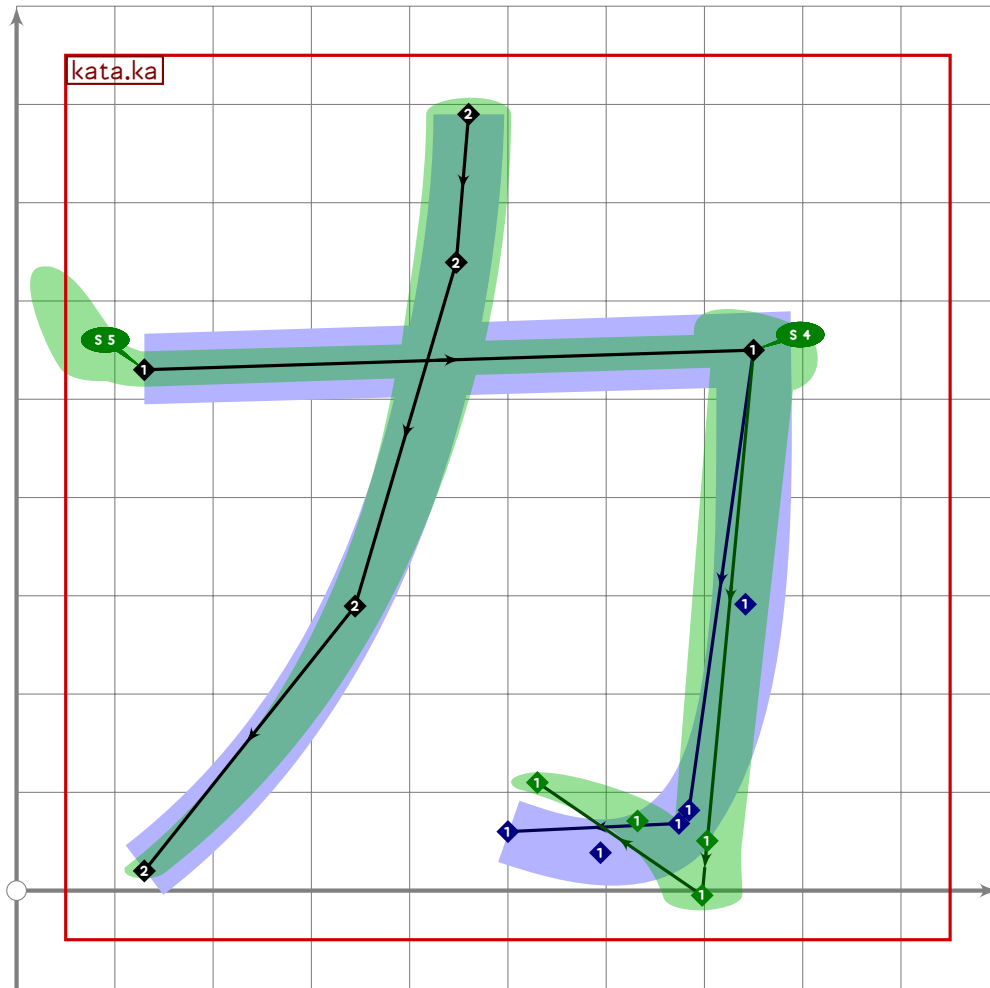
97
98 vardef kata.o =
99   push_pbox_toexpand("kata.o");
100
101   push_stroke((130+100*mincho,570)-(870-90*mincho,570),
102     (1.8,1.8)-(1.6,1.6));
103   set_boserif(0,0,5);
104   set_boserif(0,1,6);
105
106   kata.ho_centre((610,790),(610,20));
107
108   kata.no_stroke((550,570),(110,130));
109   expand_pbox;
110 enddef;
111

```

**KATA**

## Katakana Kakikukeko/Gagigugego

112 %%%%%%%%% KATAKANA KAKIKUKEKO/GAGIGUGEGO

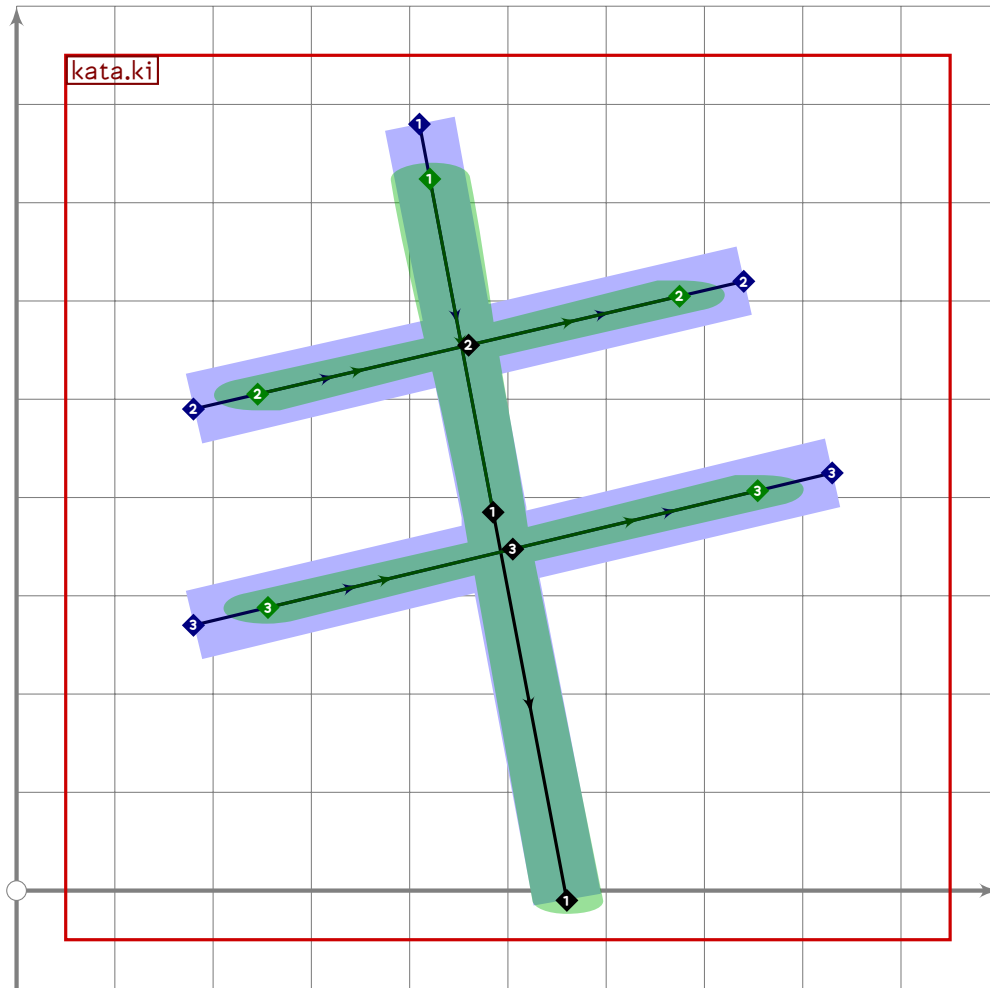


```

113
114 vardef kata.ka =
115   push_pbox_toexpand("kata.ka");
116
117   kata.ho_centre((750,550),(700,20));
118
119   replace_strokep(0)((130,530)—oldp);
120   replace_strokeq(0)((1.8,1.8)—oldq);
121   set_botip(0,1,1);
122   set_botip(0,2,whatever);
123   set_botip(0,3,0);
124   set_boserif(0,0,5);
125   set_boserif(0,1,4);
126
127   kata.no_stroke((460,790),(130,20));
128   expand_pbox;
129 enddef;

```

**KATA**

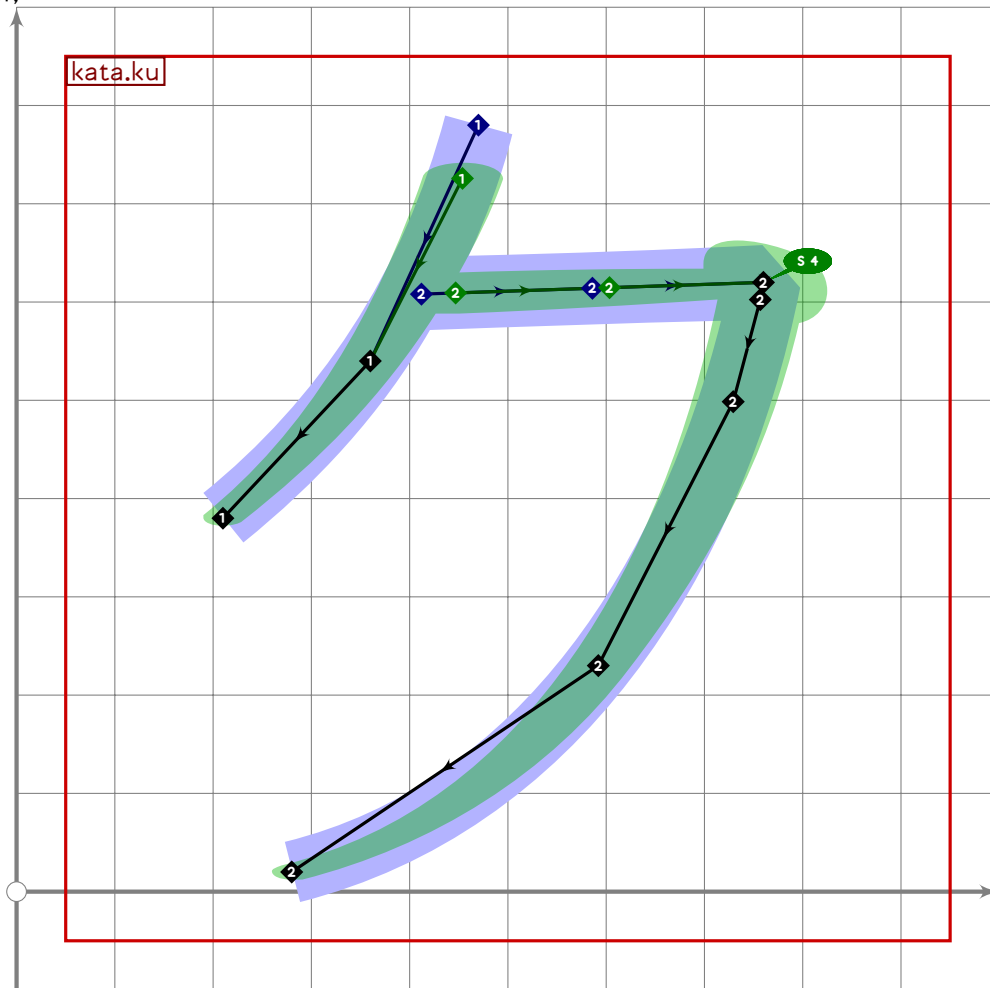


```

130
131 vardef kata.ki =
132   push_pbox_toexpand("kata.ki");
133
134   push_stroke((410,780)–(560,10),
135     (0.74,2.55)–(1.4,1.4)–(1.5,1.5));
136   replace_strokep(0)(insert_nodes(oldp)(0.5));
137   set_boserif(0,0,8);
138
139   push_stroke((180,490)–(740,620),
140     (0.6,3)–(1.6,1.6)–(0.6,3));
141   replace_strokep(0)(insert_nodes(oldp)(0.5));
142   set_boserif(0,0,5);
143   set_boserif(0,2,6);
144
145   push_stroke((180,270)–(830,425),
146     (0.6,3)–(1.6,1.6)–(0.6,3));
147   replace_strokep(0)(insert_nodes(oldp)(0.5));
148   set_boserif(0,0,5);
149   set_boserif(0,2,6);
150   expand_pbox;

```

151 endif;



152

153 vardef kata.ku =

154 push\_pbox\_toexpand("kata.ku");

155

156 push\_stroke((470,780)..(360,540)..(210,380),

157 (0.68,2.7)-(1.4,1.4)-(1.1,1.1));

158 set\_boserif(0,0,5);

159

160 z1=(get\_strokep(0) intersectionpoint ((0,600)-(1000,620)))+10\*right;

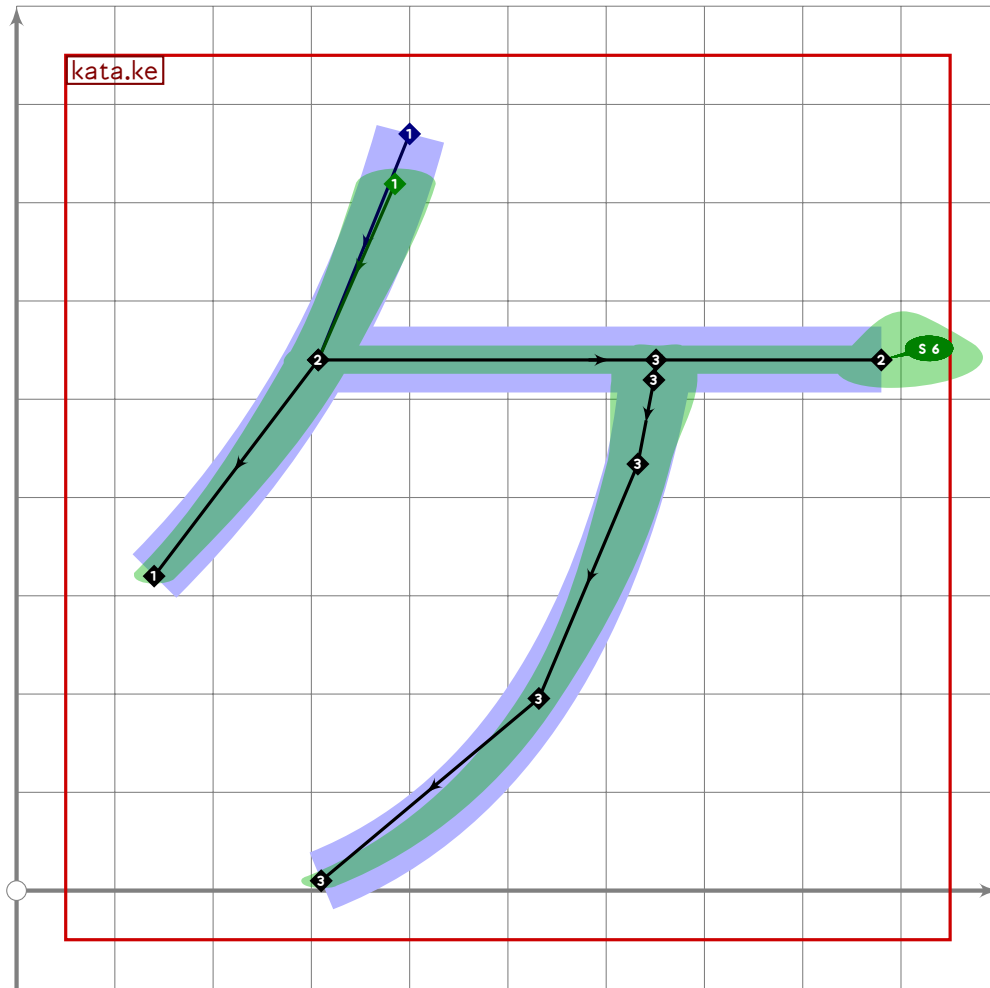
161 kata.fu\_stroke(z1,(760,620),(280,20));

162 set\_boserif(0,0,whatever);

163 expand\_pbox;

164 endif;

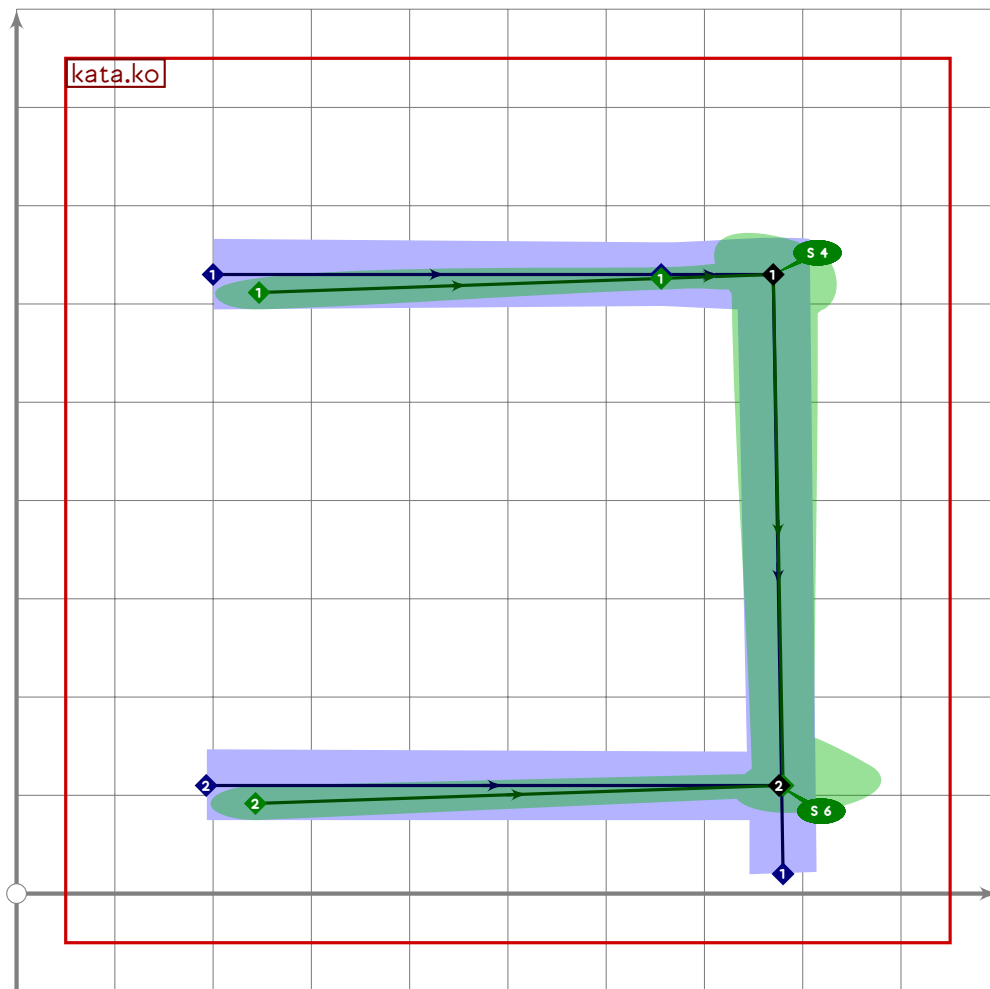
**KATA**



```

165
166 vardef kata.ke =
167   push_pbox_toexpand("kata.ke");
168
169   push_stroke((400,770)..(307,540)..(140,320),
170     (0.68,2.7)-(1.4,1.4)-(1.1,1.1));
171   set_boserif(0,0,5);
172
173   z1=(get_strokep(0) intersectionpoint ((0,540)-(1000,540)));
174   push_stroke(z1-(880,540),(1.5,1.5)-(1.5,1.5)-(0.75,2.85));
175   set_boserif(0,1,6);
176
177   kata.no_stroke(point 0.6 of (z1-(880,540)),(310,10));
178   replace_strokep(0)(insert_nodes(oldp)(0.2));
179   expand_pbox;
180 enddef;

```



```

181
182 vardef kata.ko =
183   push_pbox_toexpand("kata.ko");
184
185   push_stroke((200,630-20*mincho)-(770,630)-(780,mincho[20,110]),
186     (0.78,2.83)-(1.3,1.3)-(1.7,1.7)-(1.4,1.4));
187   replace_strokep(0)(insert_nodes(oldp)(0.8));
188   set_botip(0,2,1);
189   set_boserif(0,0,5);
190   set_boserif(0,2,4);
191
192   push_stroke((193,110-20*mincho)-(776,110),
193     (0.78,2.83)-(1.4,1.4));
194   set_boserif(0,0,5);
195   set_boserif(0,1,6);
196   expand_pbox;
197 enddef;
198

```

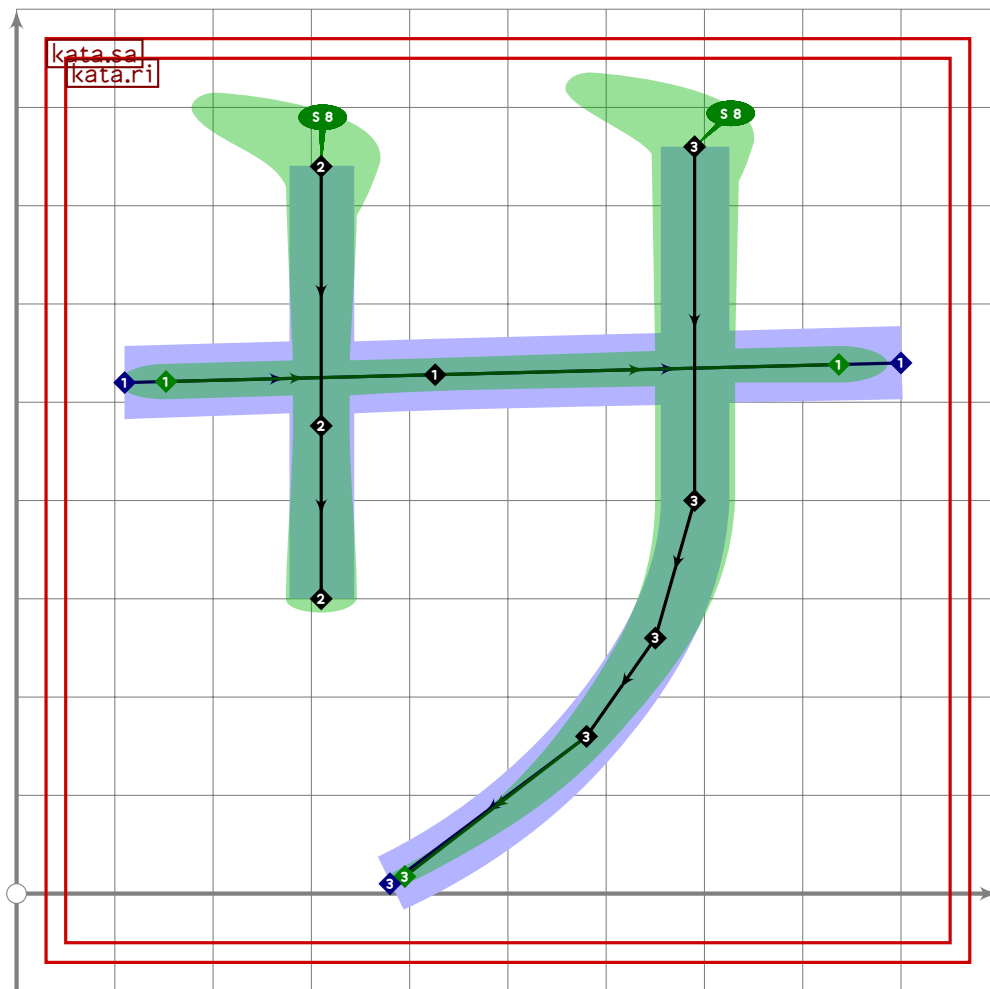
KATA

## Katakana Sashisuseso/Zajizuzezo

```

199 %%%%%%%%% KATAKANA SASHISUSES0/ZAJIZUZEZO

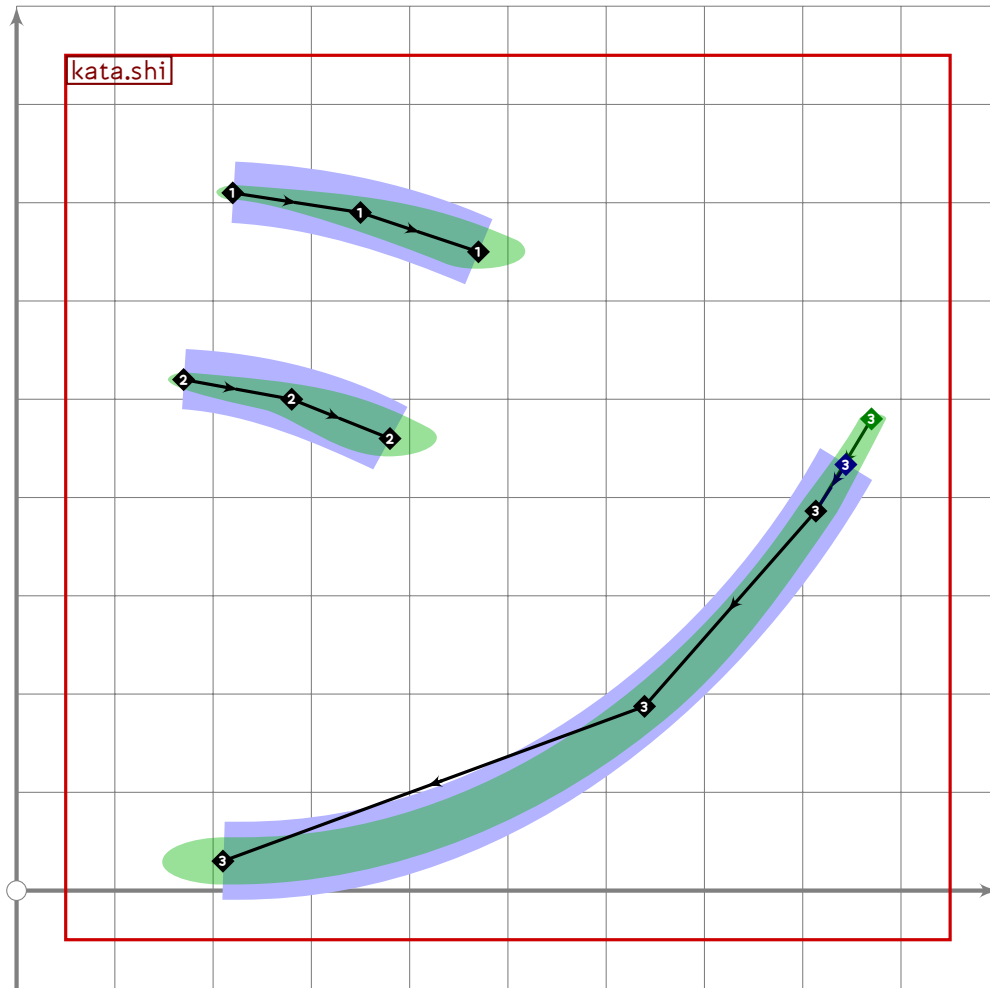
```



```

200
201 vardef kata.sa =
202   push_pbox_toexpand("kata.sa");
203
204   push_stroke((110,520)–(900,540),
205     (0.7,3)–(1.7,1.7)–(0.7,3));
206   replace_strokep(0)(insert_nodes(oldp)(0.4));
207   set_boserif(0,0,5);
208   set_boserif(0,2,6);
209
210   kata.ri;
211   expand_pbox;
212 enddef;

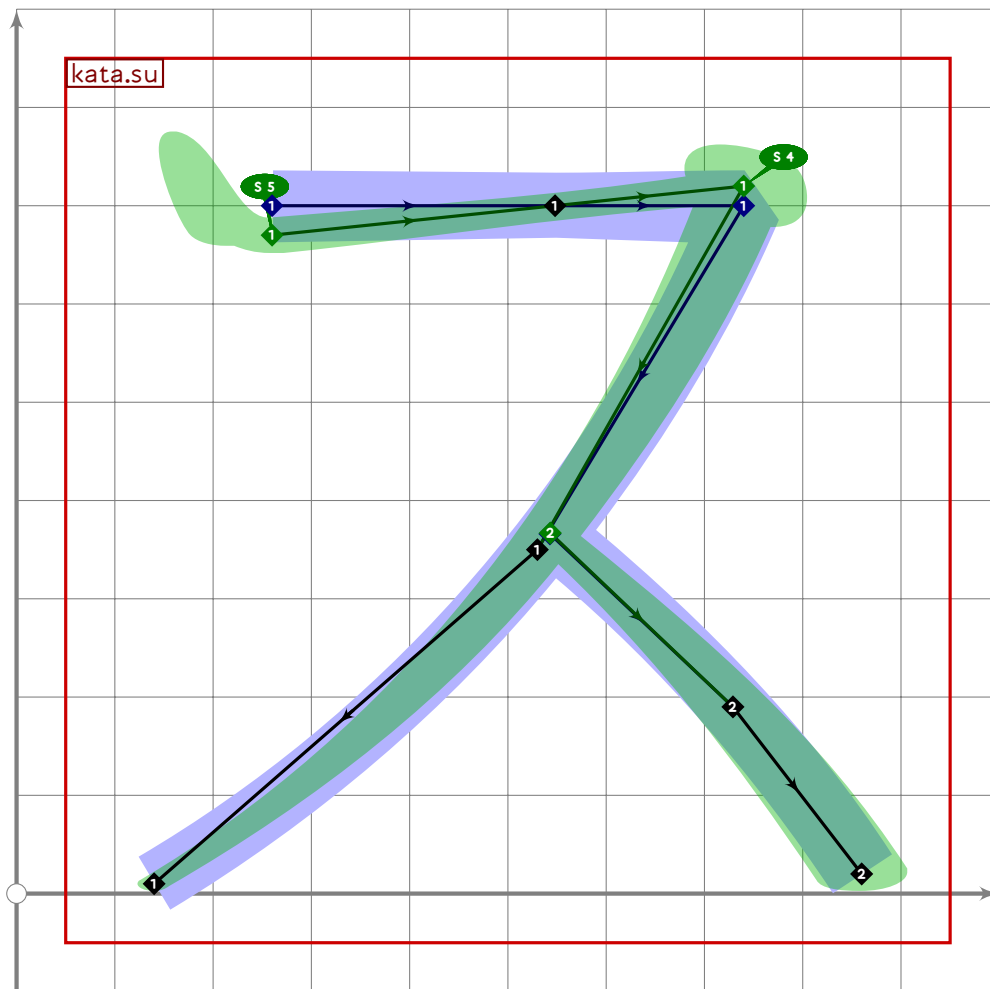
```



```

213
214 vardef kata.shi =
215   push_pbox__toexpand("kata.shi");
216
217   push_stroke((220,710)..(350,690)..(470,650),
218     (1,1)..(1.6,1.6)..(1.8,1.8));
219
220   push_stroke((170,520)..(280,500)..(380,460),
221     (1,1)..(1.6,1.6)..(1.8,1.8));
222
223   kata.no_stroke((870,480),(210,30));
224
225   replace_strokeq(0)((0.9,0.9)-(1.1,1.1)-(1.4,1.4)-(2.2,2.2));
226   set_boserif(0.4,5);
227   expand_pbox;
228 enddef;

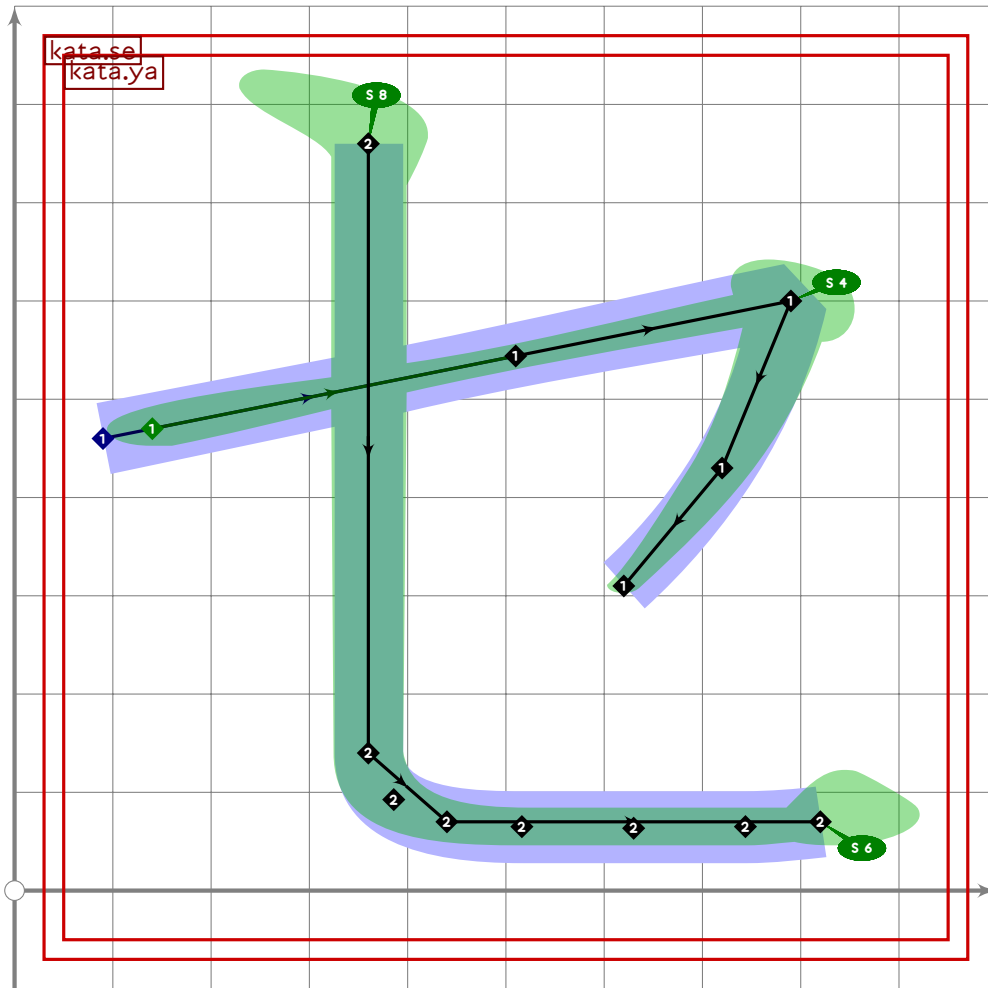
```



```

229
230 vardef kata.su =
231   push_pbox__toexpand("kata.su");
232
233   push_stroke((260,700-30*mincho)-(740,700+20*mincho)..(530,350)..(140,10),
234     (1,8,1,8)-(1,3,1,3)-(1,7,1,7)-(1,4,1,4)-(1,1));
235   replace_strokep(0)(insert_nodes(oldp)(0,6));
236   set_botip(0,2,0);
237   set_boserif(0,0,5);
238   set_boserif(0,2,4);
239
240   push_stroke((point 2.95 of get_strokep(0))..(729,190)..(860,20),
241     (1,2,1,2)-(1,6,1,6)-(1,8,1,8));
242   expand_pbox;
243 enddef;

```

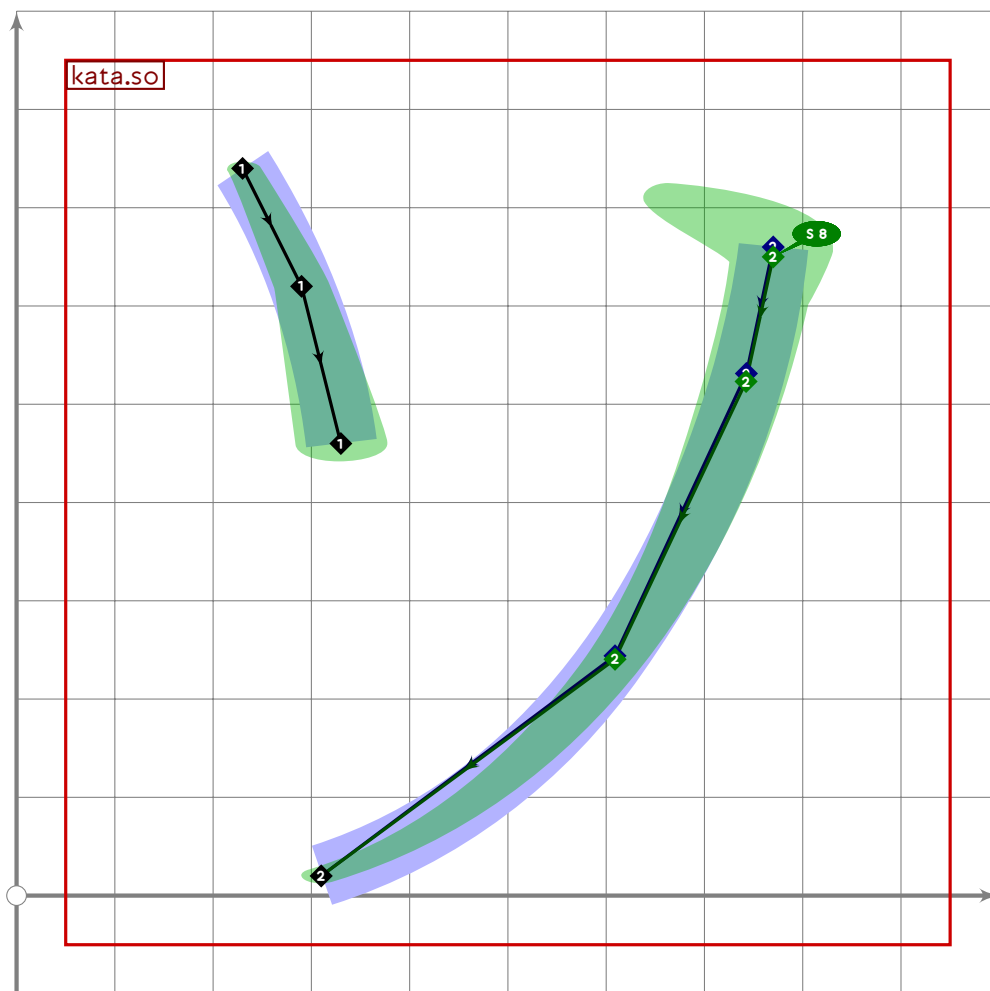


```

244
245 vardef kata.se =
246   push_pbox_toexpand("kata.se");
247
248   kata.ya;
249
250   replace_stroke(-1)(oldp shifted (-30,0));
251
252   replace_stroke(0)((360,760)-(360,140){dir 274}..
253     (440,70)..tension 2.1..(820,70));
254   replace_strokeq(0)((1.6,1.6)-(1.5,1.5)-(1.9,1.9)-(1.8,1.8));
255   set_boserif(0,0,8);
256   set_boserif(0,3,6);
257   expand_pbox;
258 enddef;

```

**KATA**



```

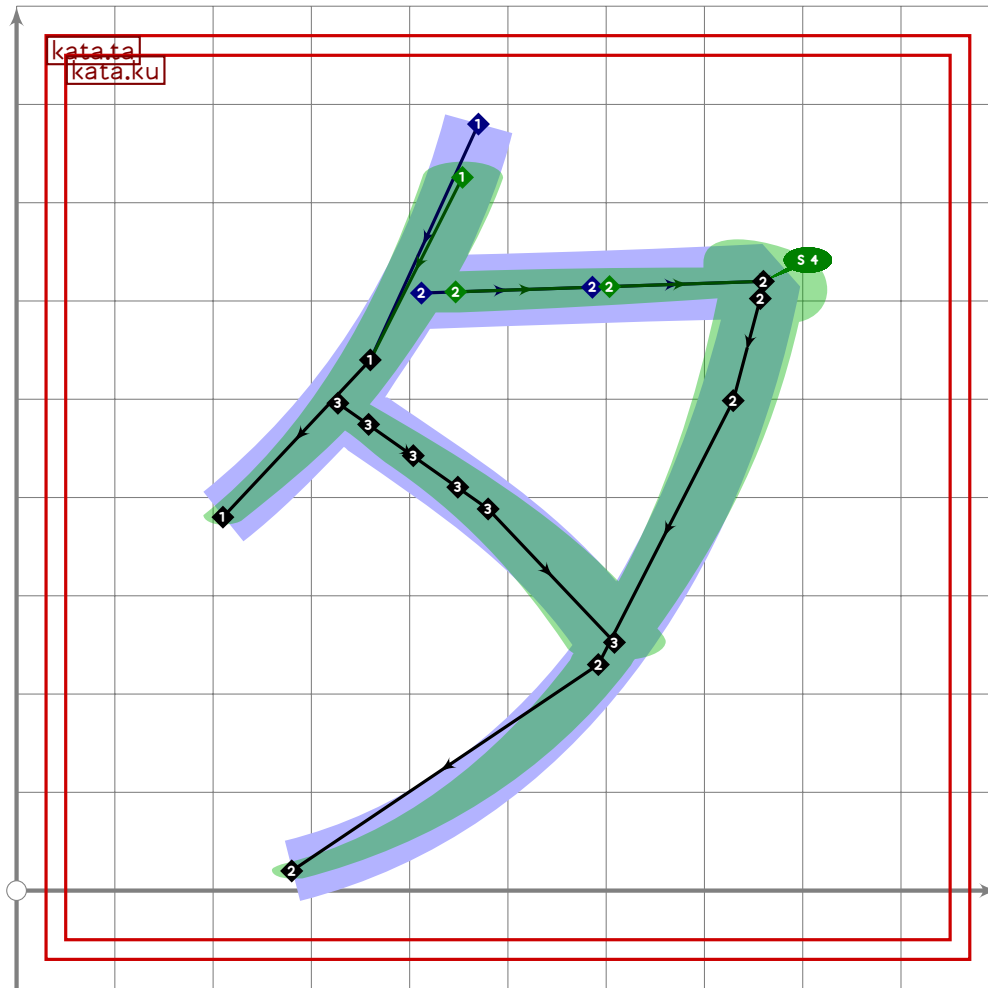
259
260 vardef kata.so =
261   push_pbox__toexpand("kata.so");
262
263   push_stroke((230,740)..(290,620)..(330,460),
264     (1,1)..(1.3,1.3)..(1.8,1.8));
265
266   kata.no_stroke((770,660-10*mincho),(310,20));
267   set_boserif(0,0,8);
268   expand_pbox;
269 enddef;
270

```

KATA

## Katakana Tachitsuteto/Dajizudedo

271 %%%%%%%%% KATAKANA TACHITSUTETO/DAJIZUDED0

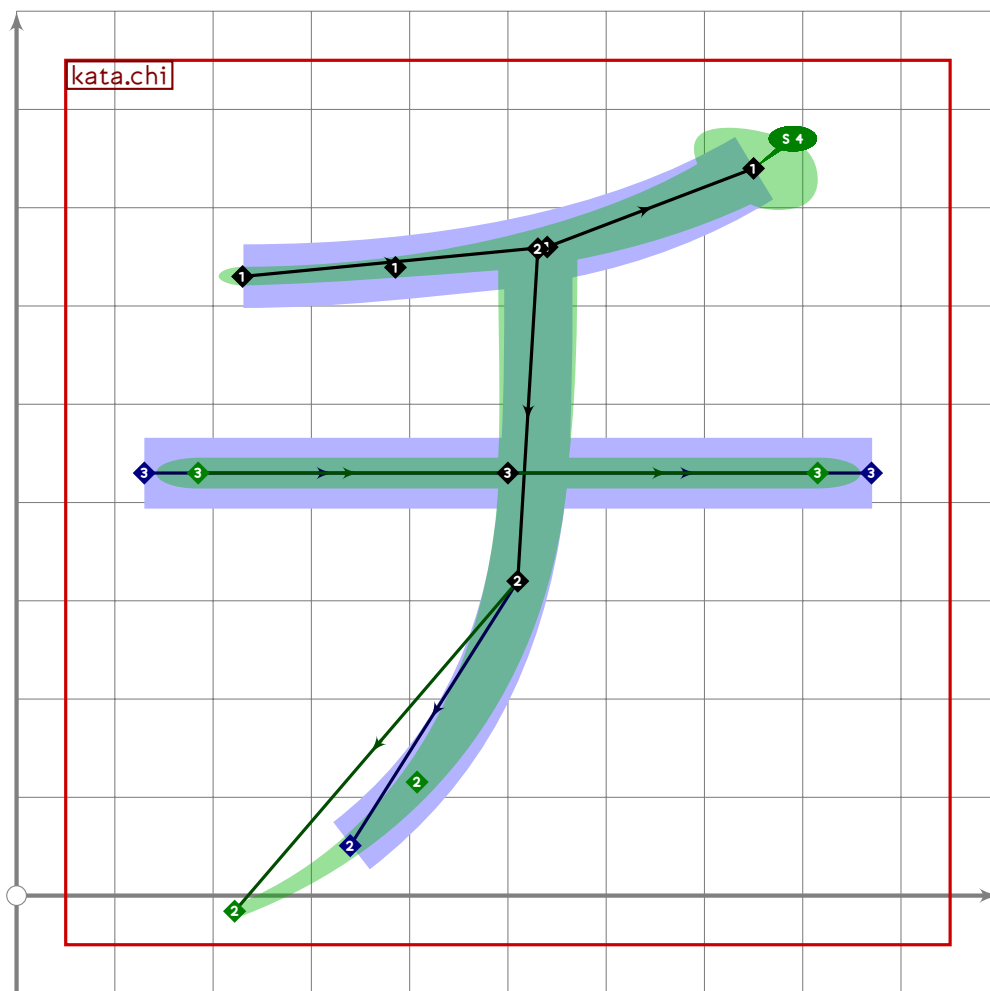


```

272
273 vardef kata.ta =
274   push_pbox_toexpand("kata.ta");
275
276   kata.ku;
277
278   numeric x[],y[];
279   z1=point 1.25 of get_stroke(-1);
280   z3=point 4.9 of get_stroke(0);
281   z2=(0.5[z1,z3])+0.05*((z3-z1) rotated 90);
282   push_stroke(z1..tension 2..z2..z3,
283     (1.2,1.2)-(1.6,1.6)-(1.9,1.9));
284   expand_pbox;
285 enddef;

```

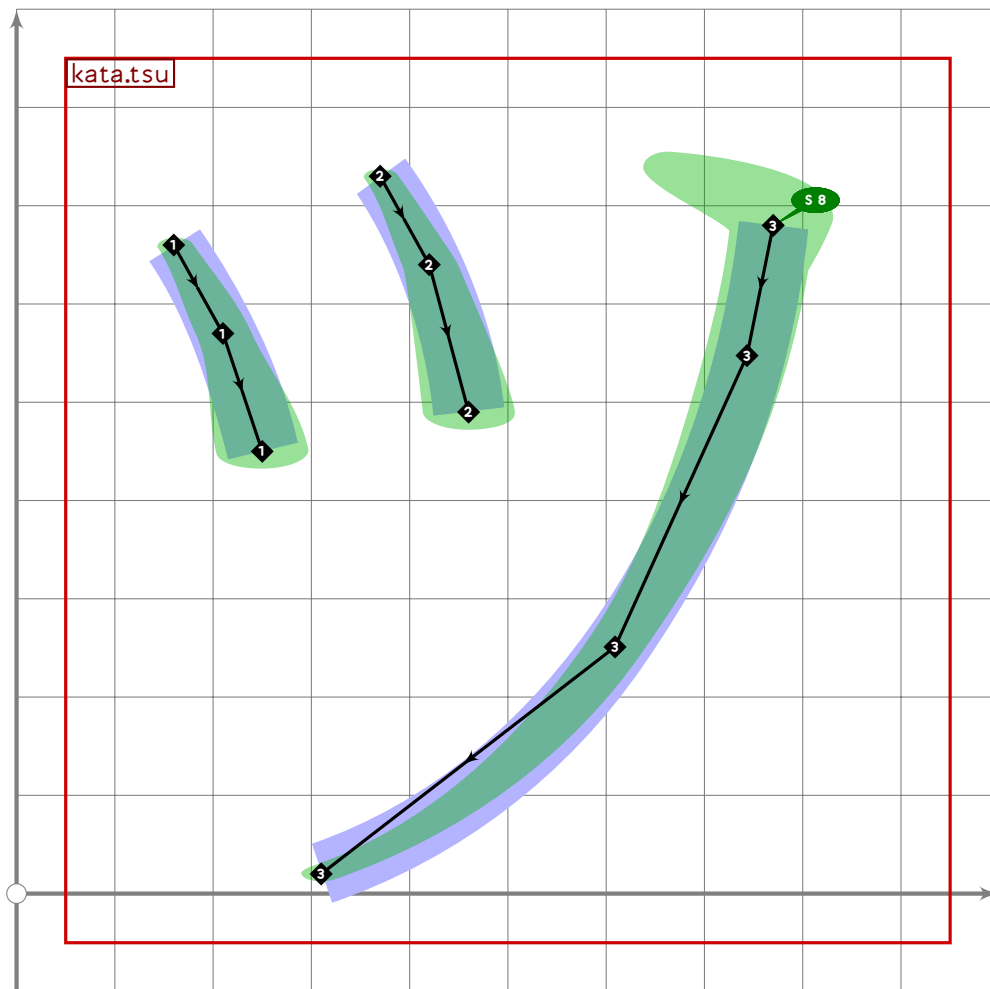
**KATA**



```

286
287 vardef kata.chi =
288   push_pbox_toexpand("kata.chi");
289
290   push_stroke((230,630)..tension 1.3..(540,660)..(750,740),
291     (1,2,1,2)-(1,7,1,7)-(2,2));
292   set_boserif(0,2,4);
293
294   kata.na_centre;
295   replace_stroke(0)(subpath (xpart (oldp intersectiontimes
296     get_stroke(-1)),infinity) of oldp);
297
298   push_stroke((130,430)-(870,430),
299     (0,7,2,7)-(1,6,1,6)-(0,7,2,7));
300   replace_stroke(0)(insert_nodes(oldp)(0.5));
301   set_boserif(0,0,5);
302   set_boserif(0,2,6);
303   expand_pbox;
304 enddef;

```



```

305
306 vardef kata.tsu =
307   push_pbox_toexpand("kata.tsu");
308
309   push_stroke((160,660)..(210,570)..(250,450),
310     (1,1)..(1.3,1.3)..(1.8,1.8));
311
312   push_stroke((370,730)..(420,640)..(460,490),
313     (1,1)..(1.3,1.3)..(1.8,1.8));
314
315   kata.no_stroke((770,680),(310,20));
316   set_boserif(0,0,8);
317   expand_pbox;
318 enddef;
319
320 vardef kata.te_top =
321   push_pbox_toexpand("kata.te_top");
322
323   push_stroke((220,690-10*mincho)-(780,690+10*mincho),
324     (0.5,2.9)-(1.6,1.6)-(0.5,2.9));
325   replace_strokep(0)(insert_nodes(oldp)(0.5));

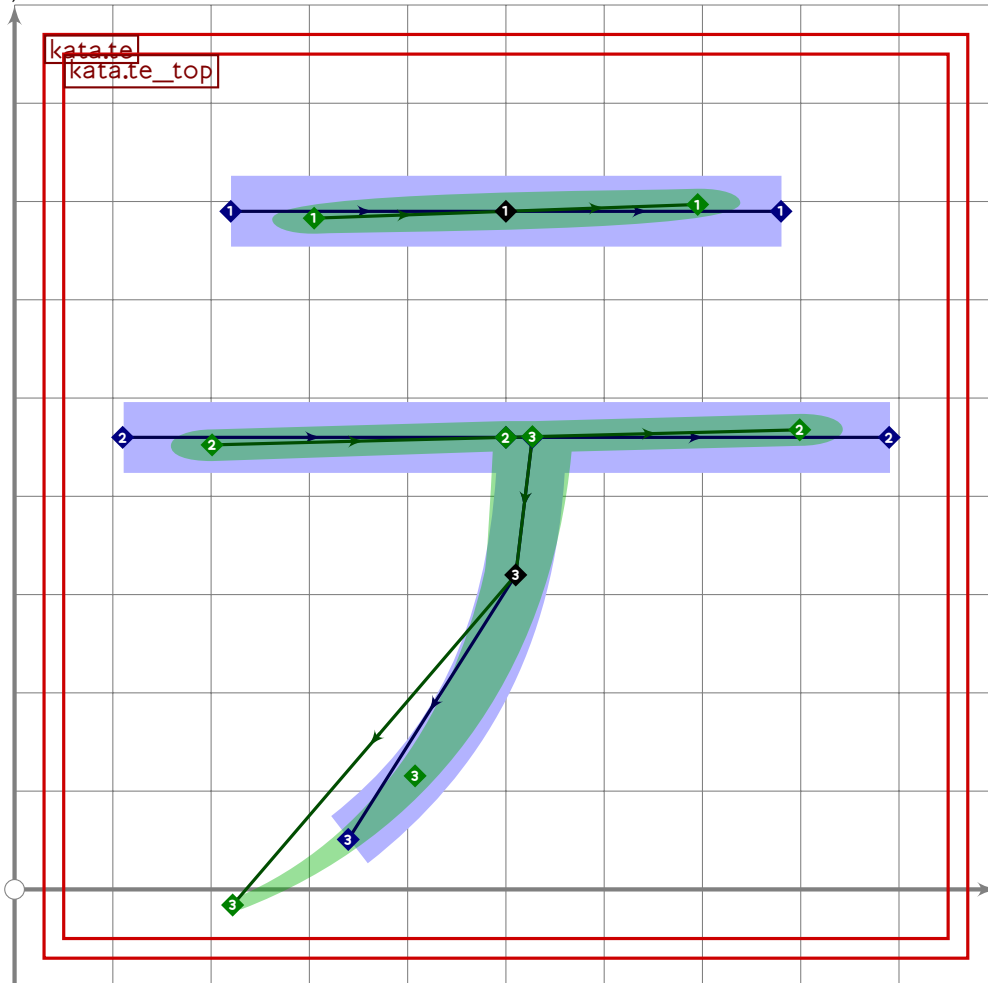
```

U+30C6  
tsuku.uni30C6

```

326 set_boserif(0,0,5);
327 set_boserif(0,2,6);
328
329 push_stroke((110,460-10*mincho)-(890,460+10*mincho),
330   (0.6,2.8)-(1.6,1.6)-(0.6,2.8));
331 replace_strokep(0)(insert_nodes(oldp)(0.5));
332 set_boserif(0,0,5);
333 set_boserif(0,2,6);
334 expand_pbox;
335 enddef;

```

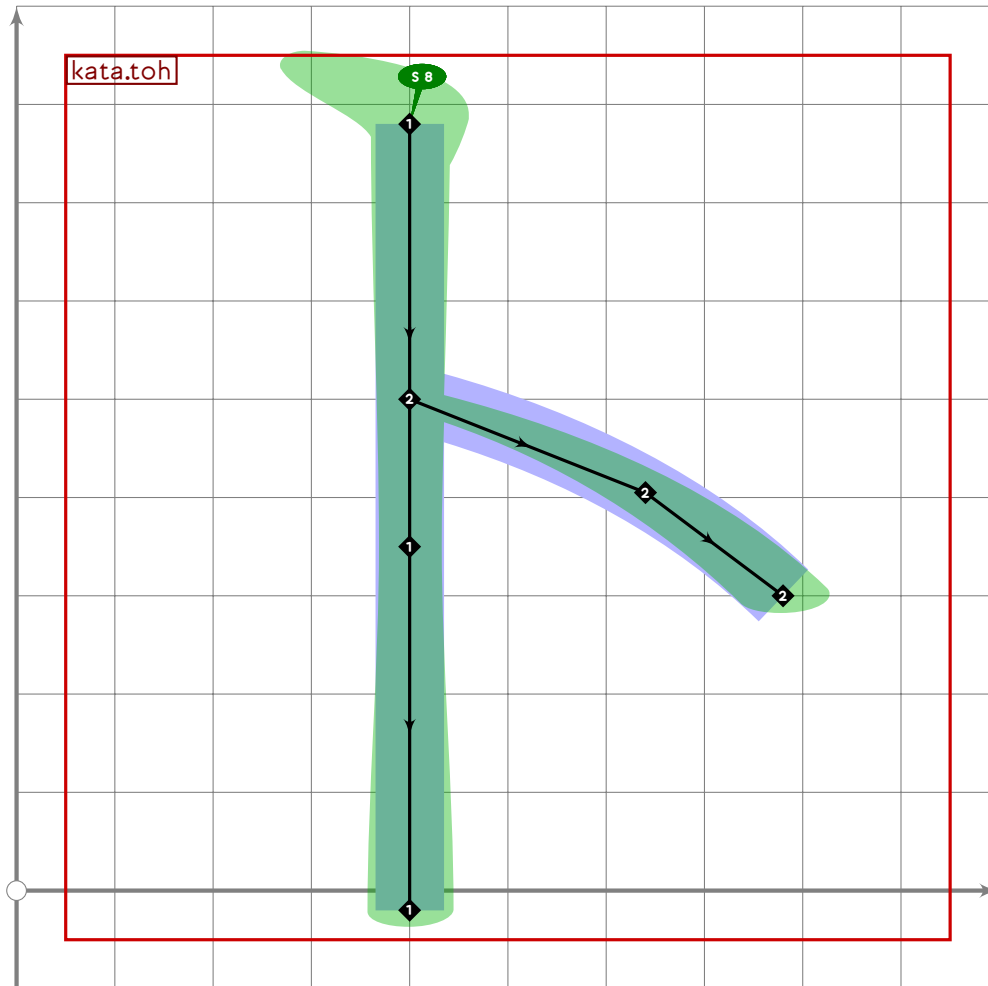


```

336
337 vardef kata.te =
338   push_pbox_toexpand("kata.te");
339
340   kata.te_top;
341
342   kata.te_bottom;
343   replace_strokep(0)(subpath (xpart (oldp intersectiontimes get_strokep(-1)),
344     infinity) of oldp);
345   expand_pbox;
346 enddef;

```

**KATA**



```

347
348 vardef kata.toh =
349   push_pbox_toexpand("kata.toh");
350
351   push_stroke((400,780)–(400,350)–(400,-20),
352     (1.6,1.6)–(1.4,1.4)–(1.7,1.7));
353   set_boserif(0,0,8);
354
355   push_stroke((400,500)..tension 1.1..(640,405)..(780,300),
356     (1.3,1.3)–(1.6,1.6)–(1.8,1.8));
357   expand_pbox;
358 enddef;
359

```

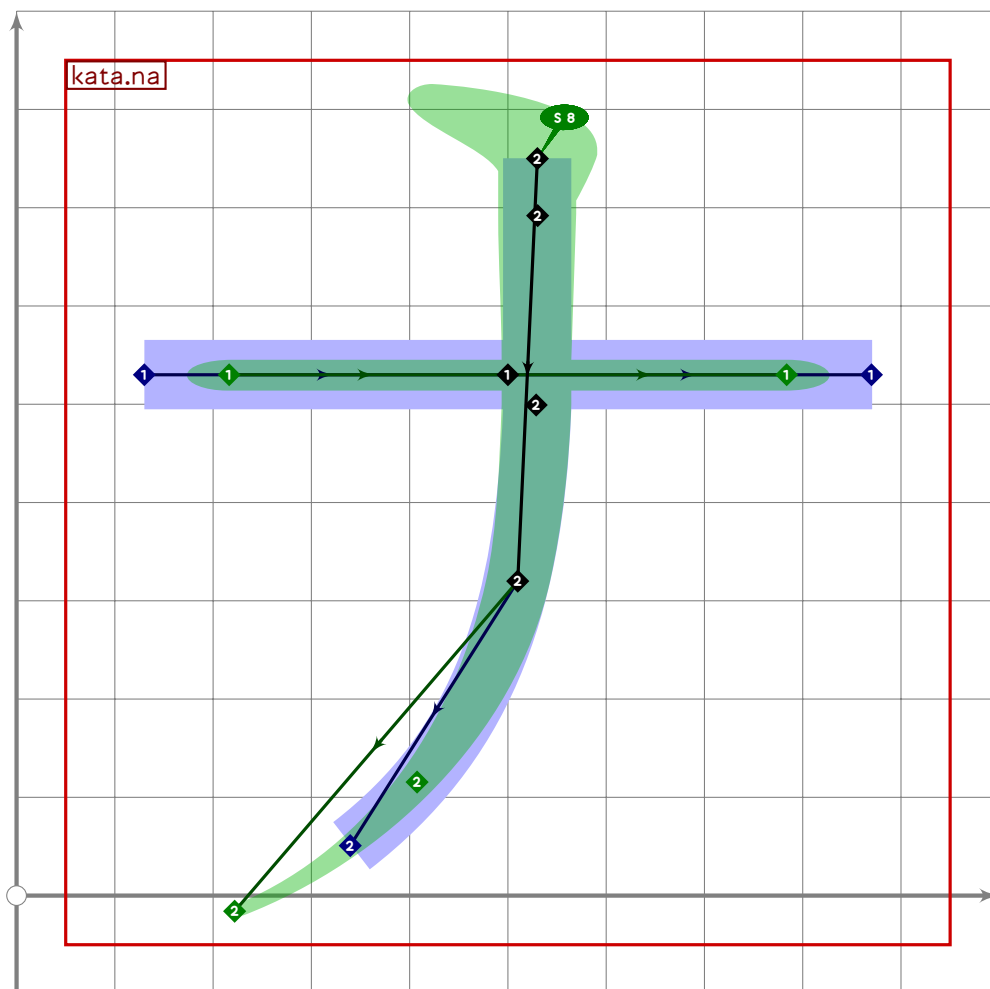
**KATA**

## Katakana Naninuneno

```

360 %%%%%%%%% KATAKANA NANINUNENO
361
362 vardef kata.na_centre =
363   push_stroke((530,750){down}..tension 1.2..(510,320)..(180,-30),
364     (1.6,1.6)–(1.4,1.4)–(0.78,0.78));
365 enddef;

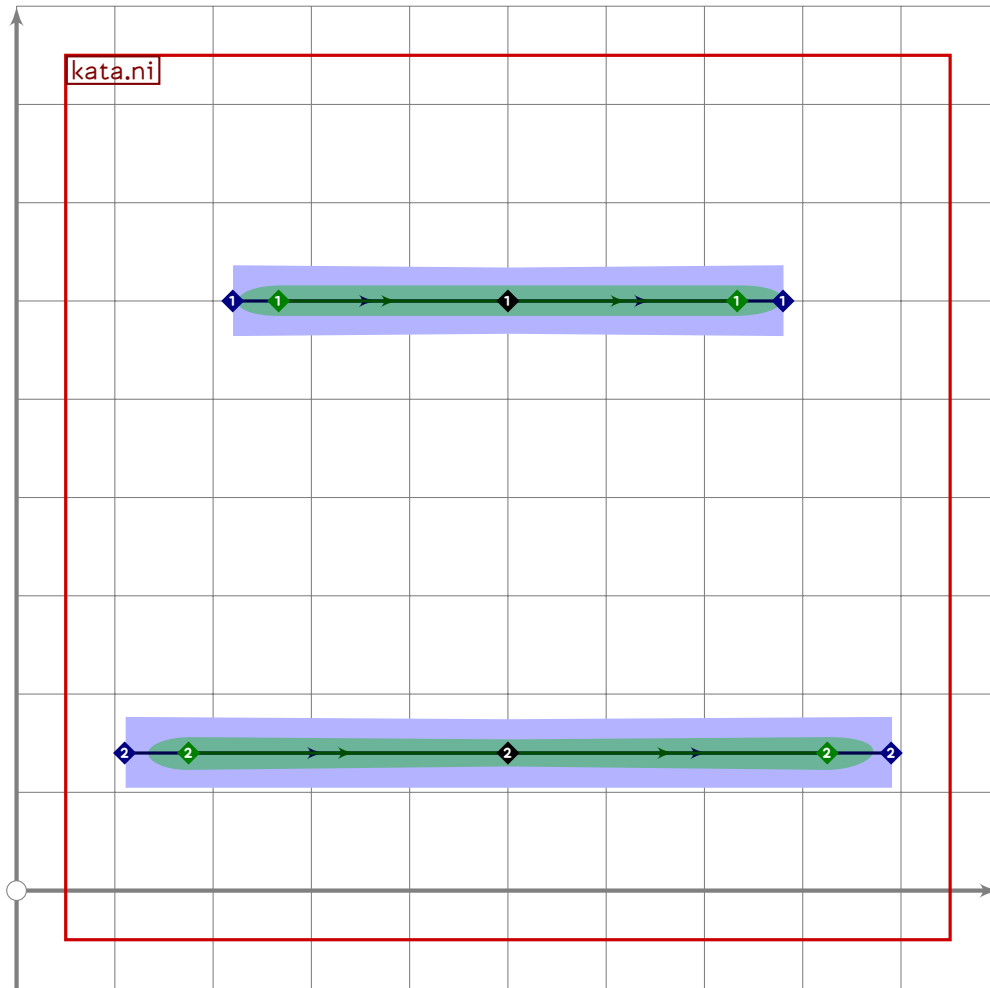
```



```

366
367 vardef kata.na =
368   push_pbox_toexpand("kata.na");
369
370   push_stroke((130,530)–(870,530),
371     (0.6,2.8)–(1.6,1.6)–(0.6,2.8));
372   replace_strokep(0)(insert_nodes(oldp)(0.5));
373   set_boserif(0,0,5);
374   set_boserif(0,2,6);
375
376   kata.na_centre;
377   set_boserif(0,0,8);
378   expand_pbox;
379 enddef;

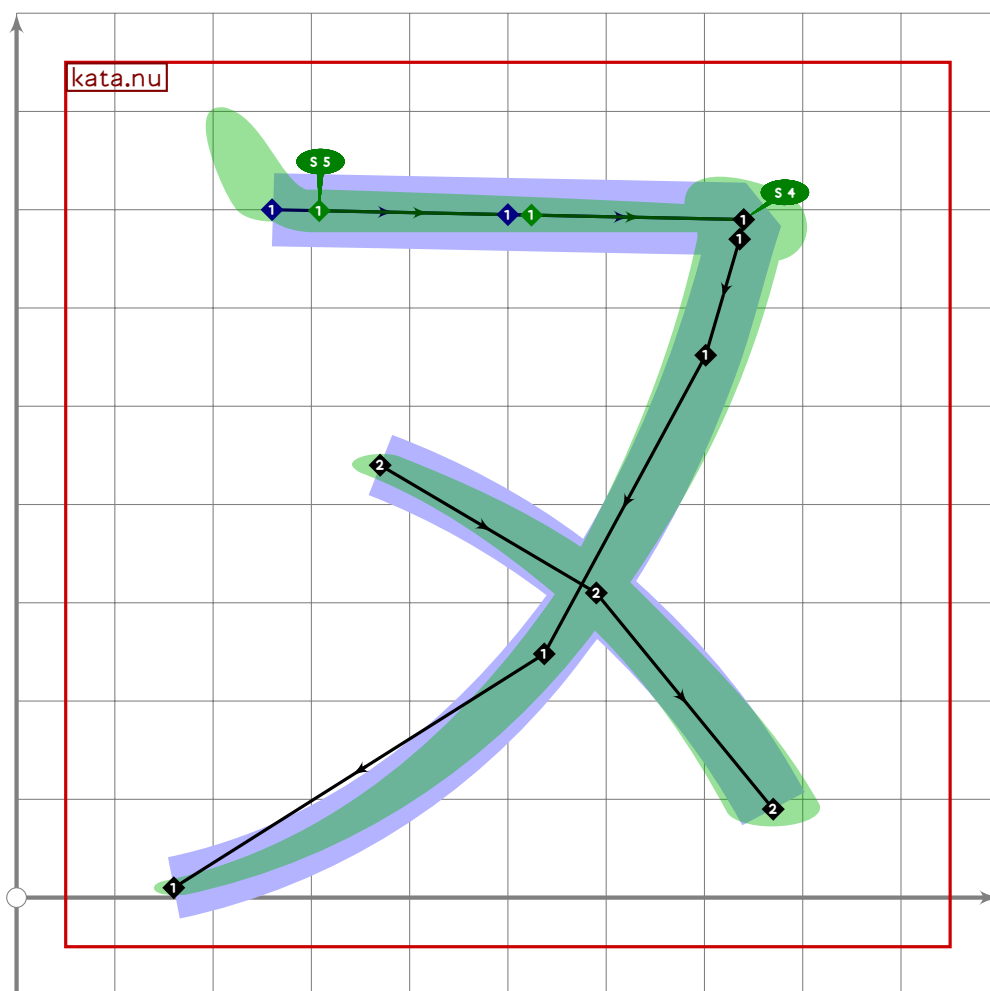
```



```

380
381 vardef kata.ni =
382   push_pbox_toexpand("kata.ni");
383
384   push_stroke((220,600)–(500,600)–(780,600),
385     (0.7,2.7)–(1.5,1.5)–(0.7,2.9));
386   set_boserif(0,0,5);
387   set_boserif(0,2,6);
388
389   push_stroke((110,140)–(500,140)–(890,140),
390     (0.7,2.7)–(1.5,1.5)–(0.7,2.9));
391   set_boserif(0,0,5);
392   set_boserif(0,2,6);
393   expand_pbox;
394 enddef;

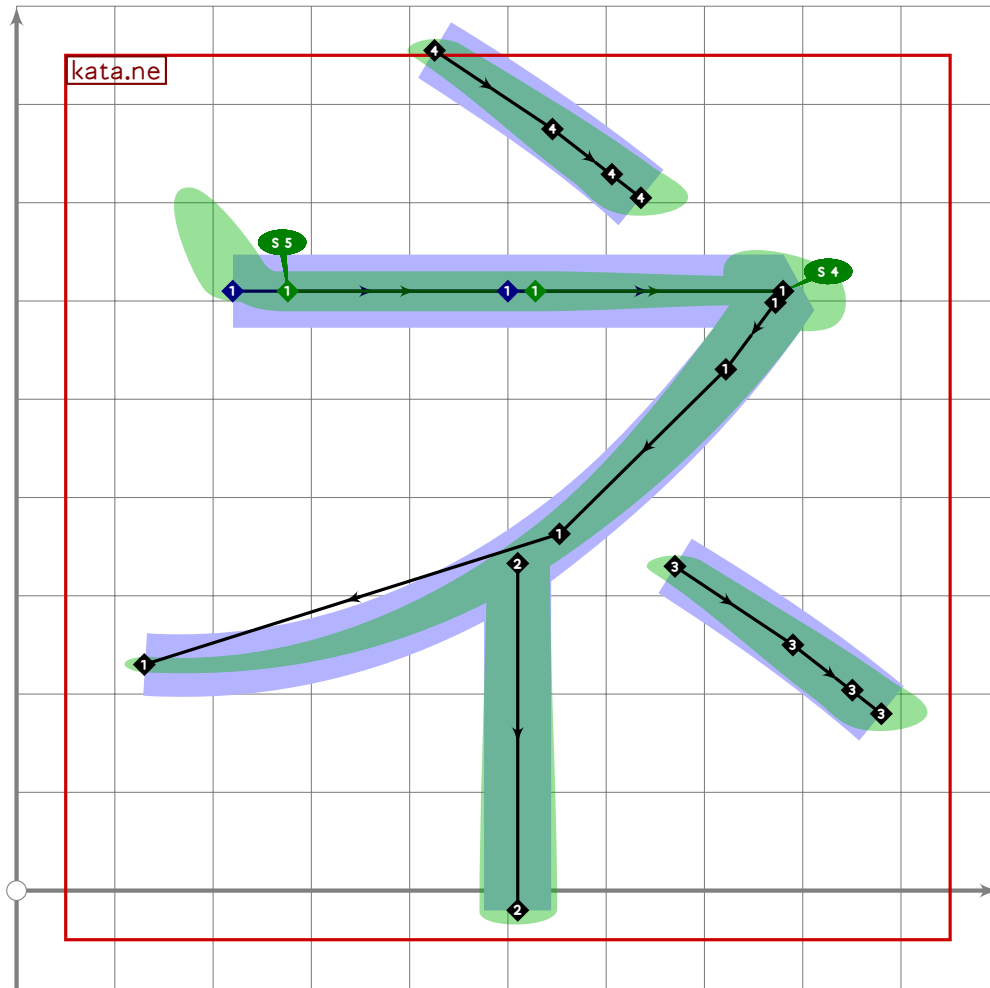
```



```

395
396 vardef kata.nu =
397   push_pbox_toexpand("kata.nu");
398
399   kata.fu_stroke((260,700),(740,690),(160,10));
400
401   push_stroke((370,440)..(590,310)..(770,90),
402     (1.3,1.3)–(1.6,1.6)–(1.8,1.8));
403   expand_pbox;
404 enddef;

```



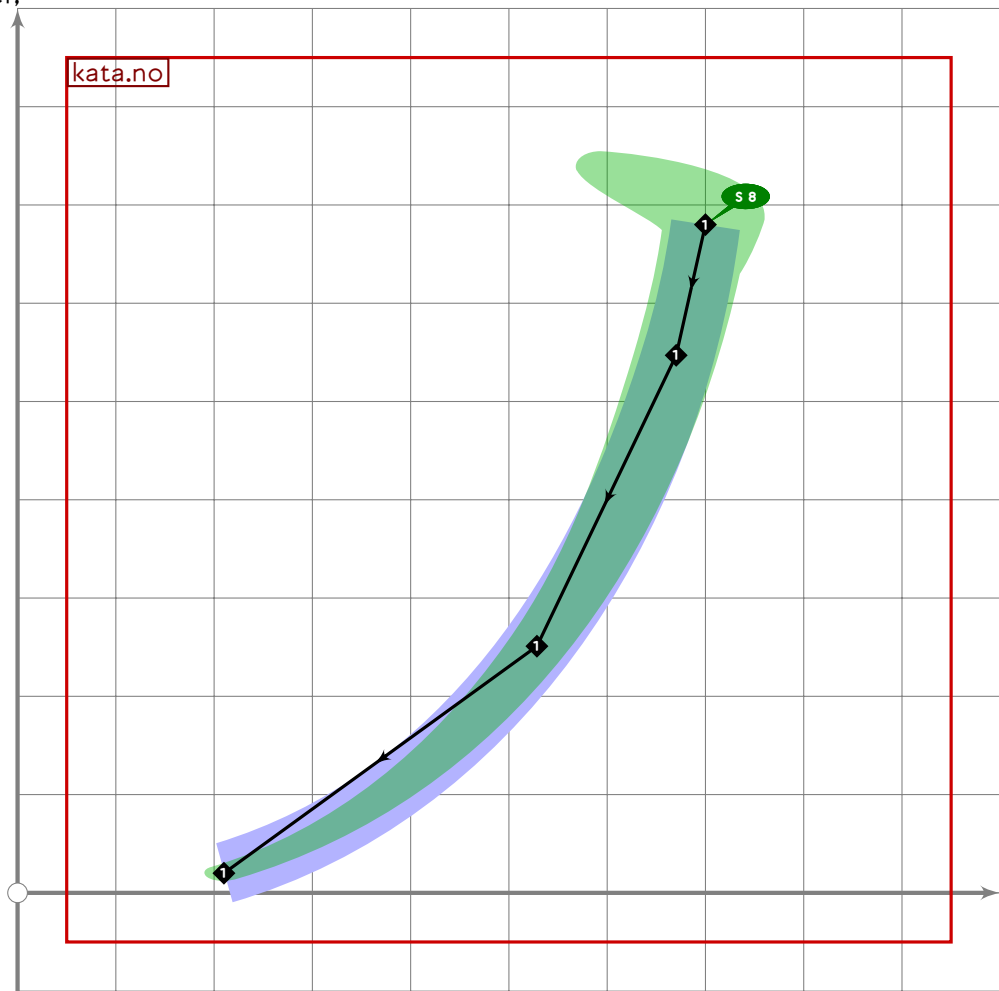
```

405
406 vardef kata.ne =
407   push_pbox_toexpand("kata.ne");
408
409   kata.fu_stroke((220,610),(780,610),(130,230));
410
411   push_stroke((((510,0)-(510,1000)) intersectionpoint
412     reverse get_stroke(0))-(510,-20),
413     (14,14)-(16,16));
414
415   push_stroke((670,330)..(790,250)..(880,180),
416     (1.3,1.3)-(1.6,1.6)-(1.8,1.8));
417
418   push_stroke(get_stroke(0) shifted ((510,800)-point 0.7 of get_stroke(0)),
419     get_stroke(0));
420   expand_pbox;
421 enddef;
422
423 vardef kata.no_stroke(expr ur,ll) =
424   push_stroke(insert_nodes(ur,tension 1.1..
425     (0.65[xpart ll,xpart ur],0.35[ypart ll,ypart ur]).{curl 1.2}ll)(0.3),

```

U+30CE  
tsuku.uni30CE

```
426 (1.7,1.7)-(1.7,1.7)-(1.4,1.4)-(1.1,1.1));  
427 enddef;
```

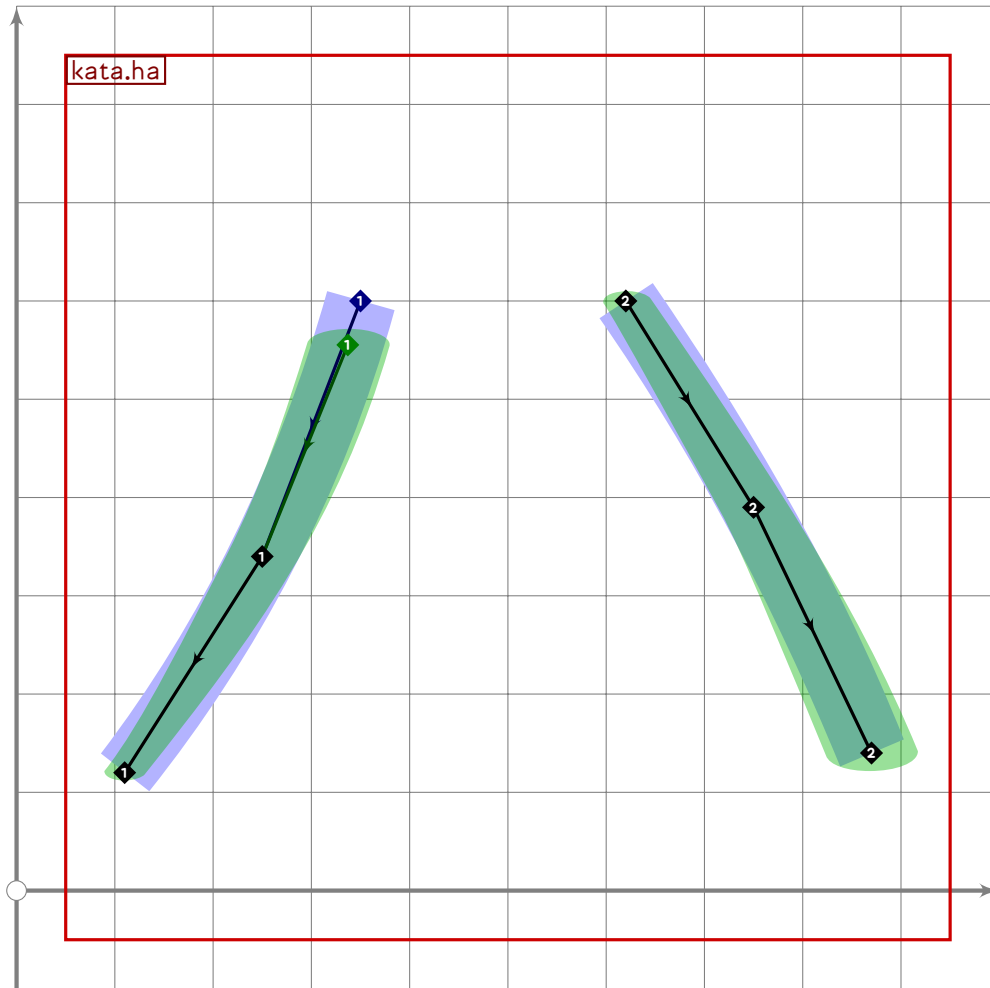


```
428  
429 vardef kata.no =  
430   push_pbox_toexpand("kata.no");  
431  
432   kata.no_stroke((700,680),(210,20));  
433   set_boserif(0,0,8);  
434   expand_pbox;  
435 enddef;  
436
```

KATA

Katakana Hahifuheho/Babibubebo/Papipupepo

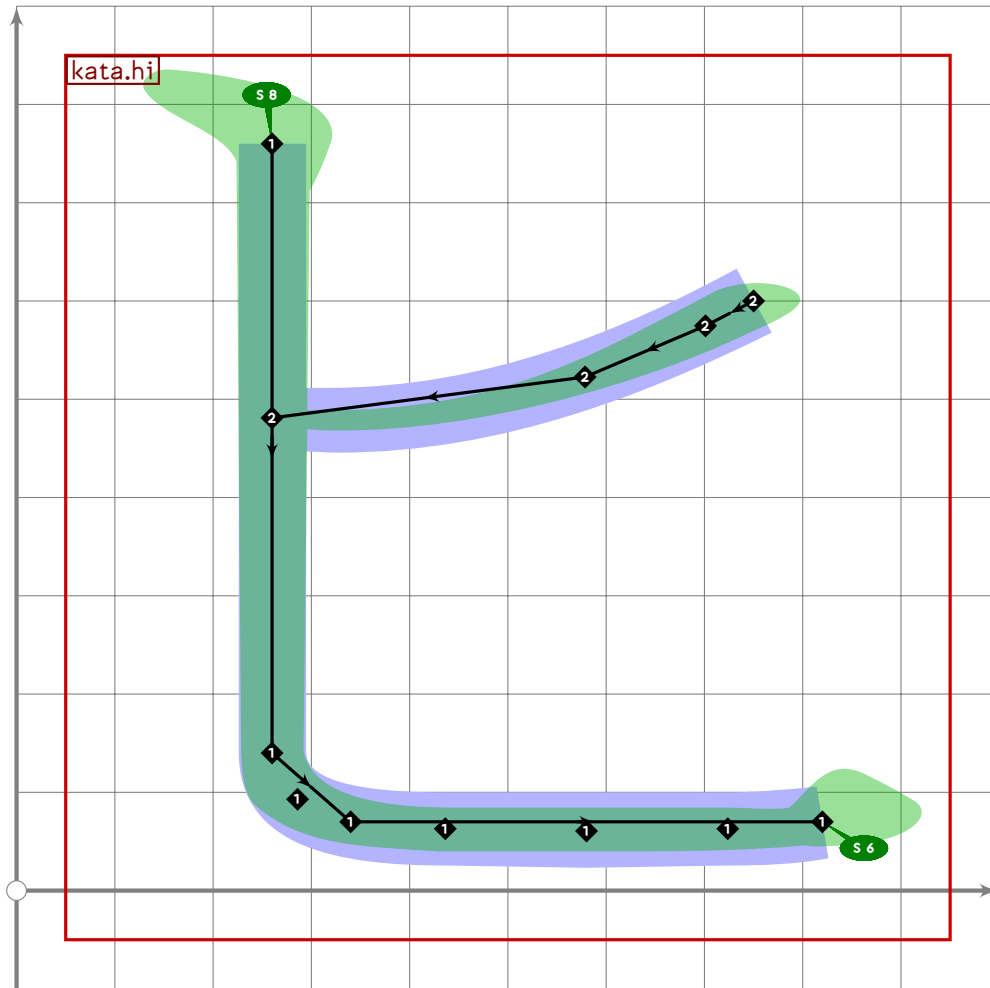
```
437 %%%%%%%%% KATAKANA HAHIFUHEHO/BABIBUBEBO/PAPIPUPEPO
```



```

438
439 vardef kata.ha =
440   push_pbox_toexpand("kata.ha");
441
442   push_stroke((350,600)..(250,340)..(110,120),
443     (0.7,2.7)-(1.5,1.5)-(1.1,1.1));
444   set_boserif(0,0,8);
445
446   push_stroke((620,600)..(750,390)..(870,140),
447     (1.2,1.2)-(1.5,1.5)-(1.8,1.8));
448   expand_pbox;
449 enddef;

```

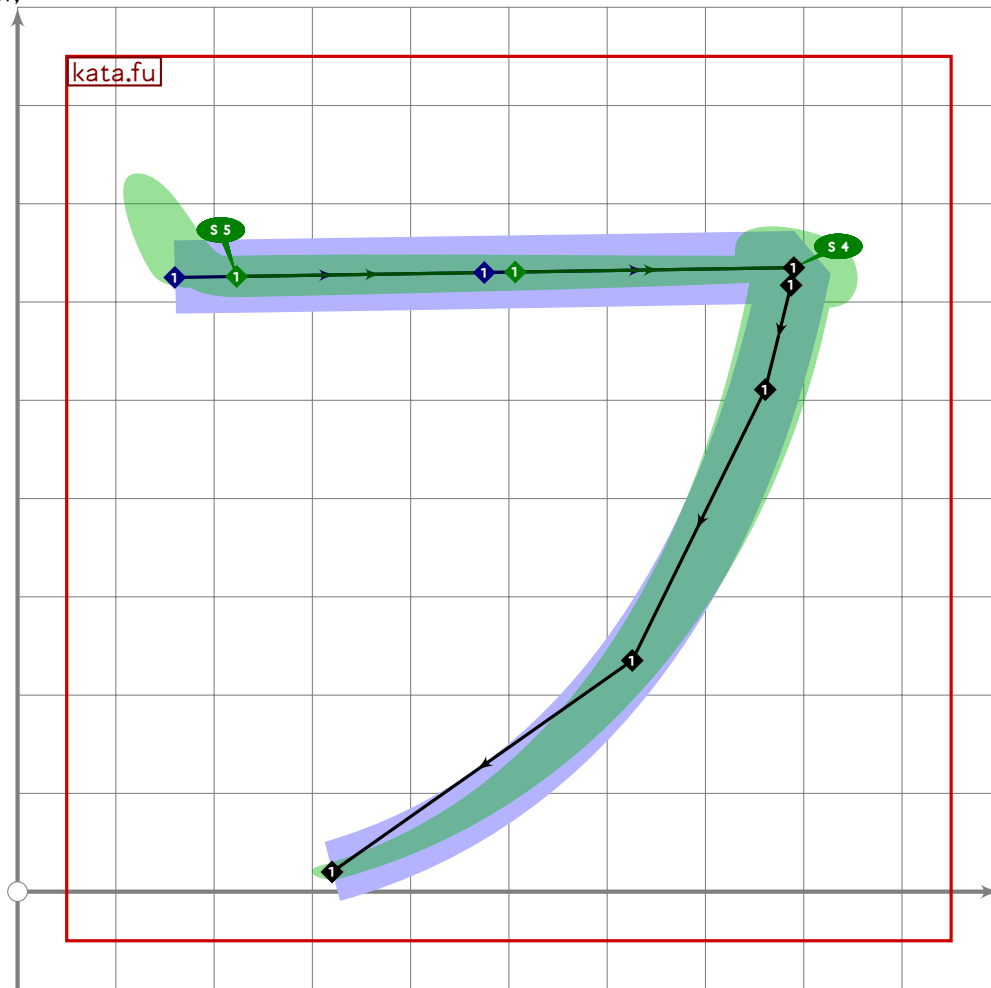


```

450
451 vardef kata.hi =
452   push_pbox_toexpand("kata.hi");
453
454   push_stroke((260,760)-(260,140){dir 274}..(340,70)..tension 2.1..(820,70),
455     (0.84,2.18)-(1.4,1.4)-(2.1,2.1)-(1.9,1.9));
456   set_boserif(0,0,8);
457   set_boserif(0,3,6);
458
459   kata.no_stroke((750,600),point 0.45 of get_stroke(0));
460   replace_strokeq(0)(oldq shifted (0,1,0.1));
461   expand_pbox;
462 enddef;
463
464 vardef kata.fu_stroke(expr ul,ur,ll) =
465   kata.no_stroke(ur,ll);
466   replace_stroke(0)(insert_nodes(((mincho*0.1)[ul,ur])-oldp)(0.5,1.15));
467   replace_strokeq(0)((2,2)-(1.9,1.9)-(1.5,1.5)-oldq);
468   set_botip(0,2,0);
469   set_boserif(0,0,5);
470   set_boserif(0,2,4);

```

471 enddef;



472

473 vardef kata.fu =

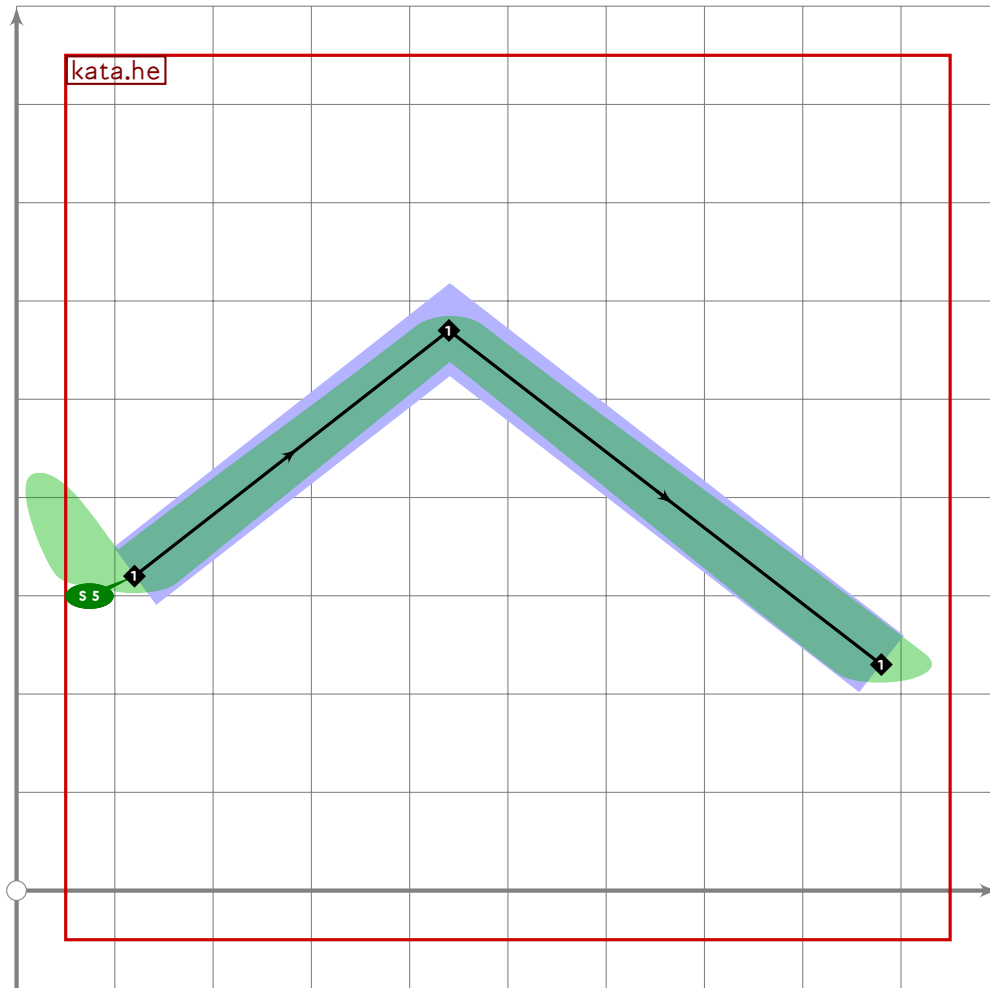
474 push\_pbox\_toexpand("kata.fu");

475

476 kata.fu\_stroke((160,625),(790,635),(320,20));

477 expand\_pbox;

478 enddef;



```

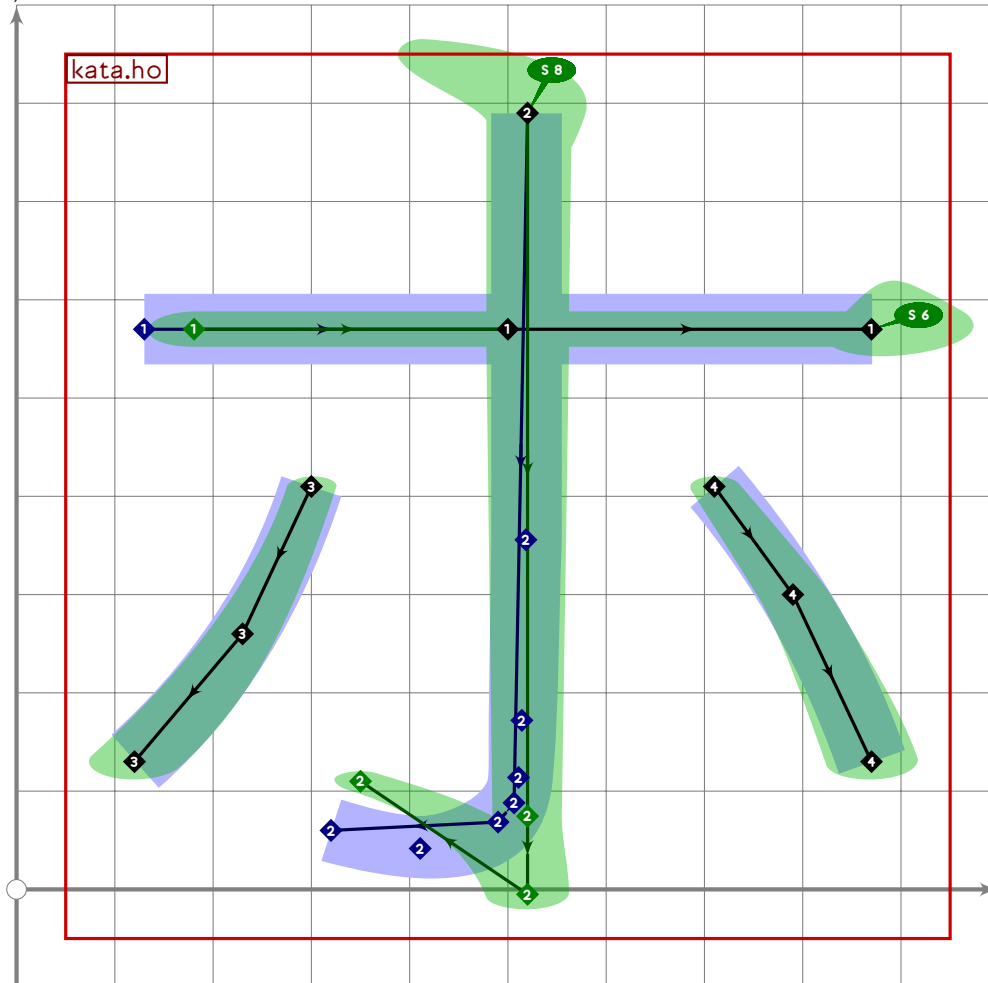
479
480 vardef kata.he =
481   push_pbox__toexpand("kata.he");
482
483   push_stroke((120,320)–(440,570)–(880,230),
484     (1.8,1.8)–(1.5,1.5)–(1.9,1.9));
485   set_botip(0,1,1);
486   set_boserif(0,0,5);
487   expand_pbox;
488 enddef;
489
490 vardef kata.ho_centre(expr pta,ptb) =
491   push_stroke(begingroup
492     numeric x[],y[];
493     path mycirc,ripx,ripy;
494     mycirc:=fullcircle scaled 100 shifted ptb;
495     z1=(pta–ptb) intersectionpoint mycirc;
496     z2=ptb+(–200,40);
497     z3=0.85[z2,z1];
498     ripx:=pta{down}..tension 1.6..z3.{curl 0}z2;
499     ripx:=pta{down}...(point 0.95 of ripx)..z3.{curl 0}z2;

```

```

500 z4=1.5[z1,ptb];
501 z5=ptb+(-170,90);
502 ripy:=pta-z4{z3-z4}..{curl 0}z5;
503 ripy:=pta-(point 0.90 of ripy)-z4{z3-z4}..{curl 0}z5;;
504 interpath(mincho,ripx,ripy)
505 endgroup,
506 (1.7,1.7)-(1.5,1.5)-(1.6,1.6)-(0.9,1.6));
507 set_botip(0,2,0);
508 set_boserif(0,0,8);
509 enddef;

```



```

510
511 vardef kata.ho =
512   push_pbox_toexpand("kata.ho");
513
514   push_stroke((130,570)-(500,570)-(870,570),
515     (0.68,2.92)-(1.8,1.8)-(1.9,1.9));
516   set_boserif(0,0,5);
517   set_boserif(0,2,6);
518
519   kata.ho_centre((520,790),(520,20));
520

```

U+30DE  
tsuku.uni30DE

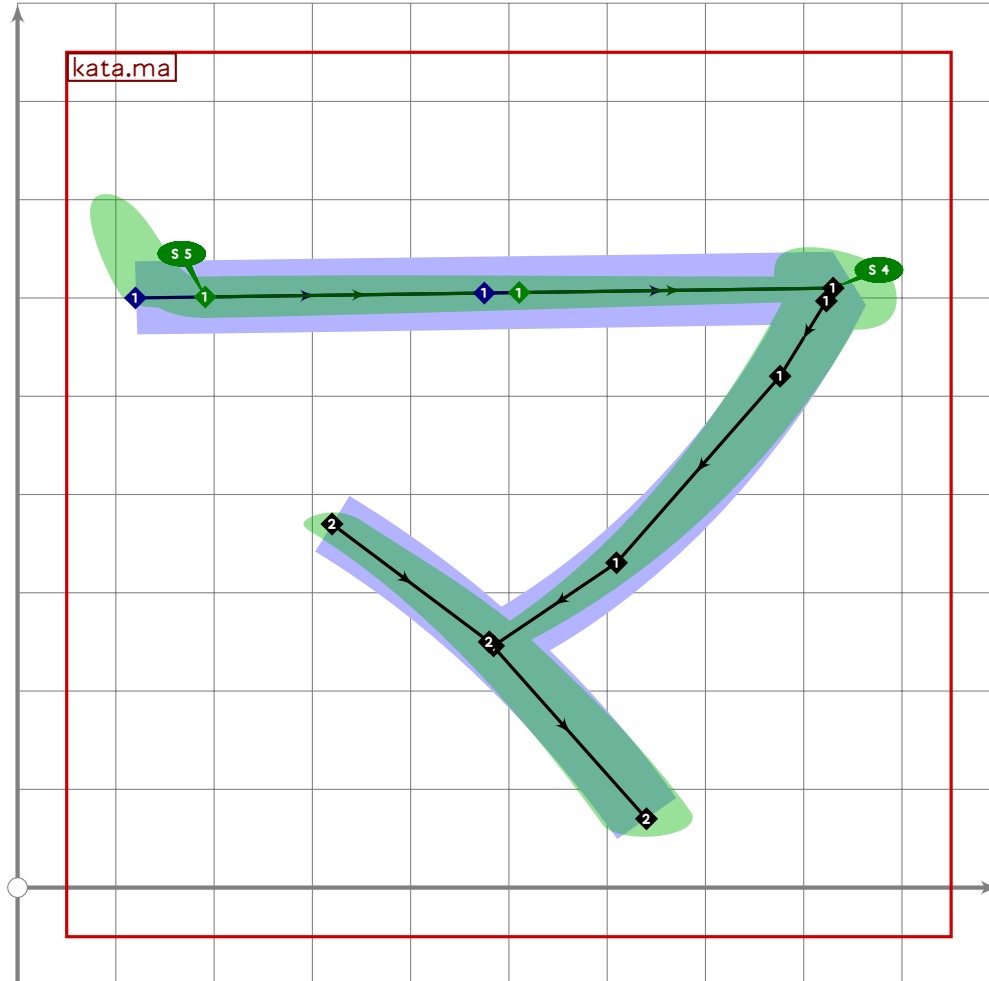
```

521  push_stroke((300,410)..(230,260)..(120,130),
522    (1.2,1.2)–(1.5,1.5)–(1.8,1.8));
523
524  push_stroke((710,410)..(790,300)..(870,130),
525    (1.2,1.2)–(1.5,1.5)–(1.8,1.8));
526  expand_pbox;
527  enddef;
528

```

## Katakana Mamimumemo

529 %%%%%%%%% KATAKANA MAMIMUMEMO



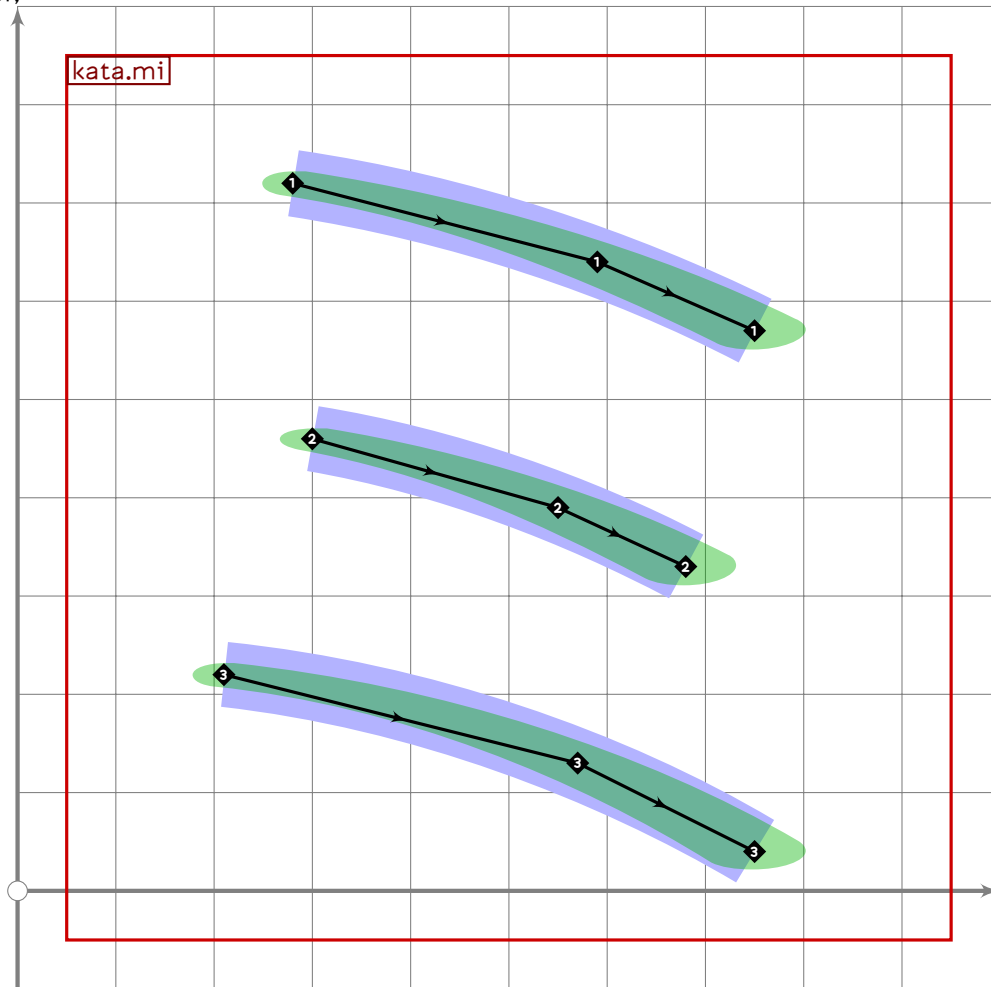
**KATA**

```

530
531  vardef kata.ma =
532    push_pbox_toexpand("kata.ma");
533
534    kata.fu_stroke((120,600),(830,610),(200,180));
535
536    push_stroke((320,370)..(480,250)..(640,70),
537      (1.3,1.3)–(1.6,1.6)–(1.8,1.8));
538    replace_stroke(-1)(subpath (0,xpart (oldp intersectiontimes
539      get_stroke(0))) of oldp);

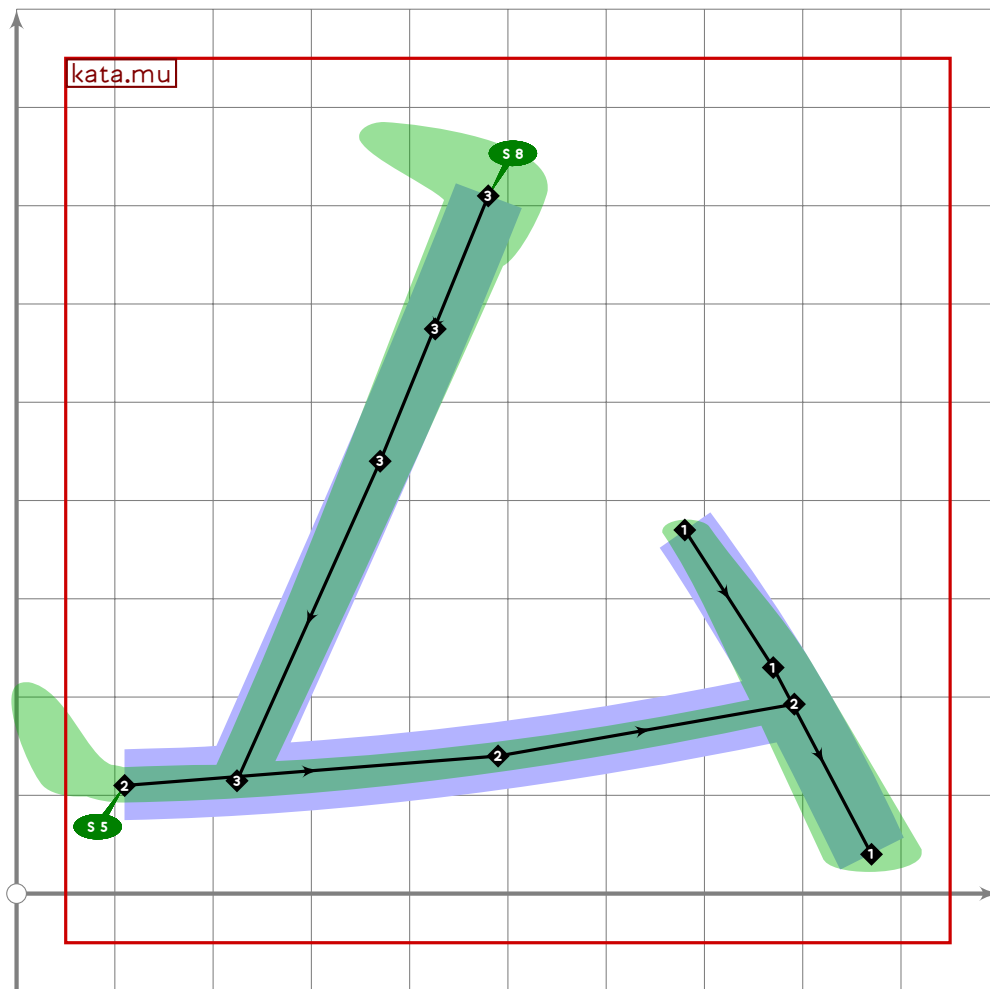
```

```
540 expand_pbox;
541 enddef;
```



```
542
543 vardef kata.mi =
544   push_pbox_toexpand("kata.mi");
545
546   push_stroke((280,720)..(590,640)..(750,570),
547     (14,14)–(17,17)–(19,19));
548
549   push_stroke((300,460)..(550,390)..(680,330),
550     (14,14)–(17,17)–(19,19));
551
552   push_stroke((210,220)..(570,130)..(750,40),
553     (14,14)–(17,17)–(19,19));
554   expand_pbox;
555 enddef;
```

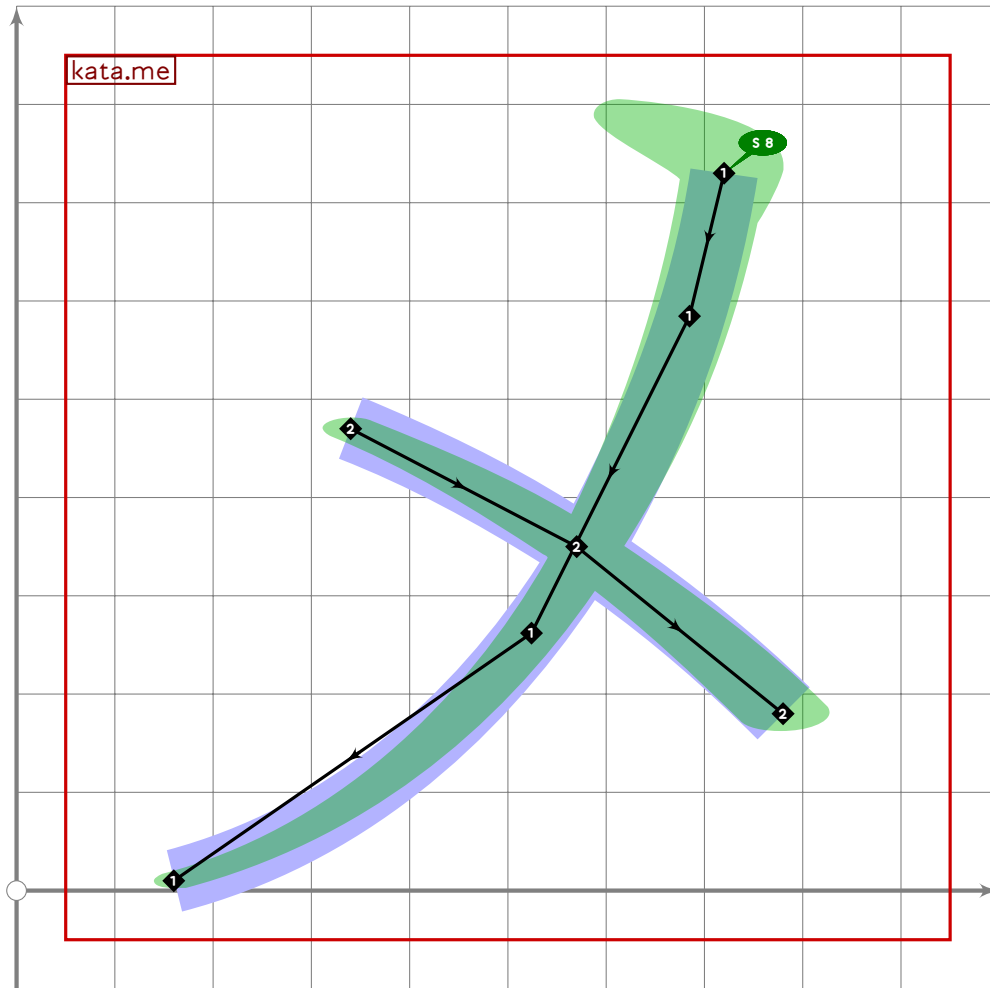
**KATA**



```

556
557 vardef kata.mu =
558   push_pbox_toexpand("kata.mu");
559
560   push_stroke((680,370)..(770,230)..(870,40),
561     (1.2,1.2)–(1.6,1.6)–(1.9,1.9));
562
563   push_stroke((110,110)..(490,140)..(point 1.2 of get_stroke(0)),
564     (1.8,1.8)–(1.6,1.6)–(1.4,1.4));
565   set_boserif(0,0,5);
566
567   push_stroke((480,710)..(370,440)..(point 0.3 of get_stroke(0)),
568     (1.7,1.7)–(1.5,1.5)–(1.3,1.3));
569   set_boserif(0,0,8);
570   expand_pbox;
571 enddef;

```

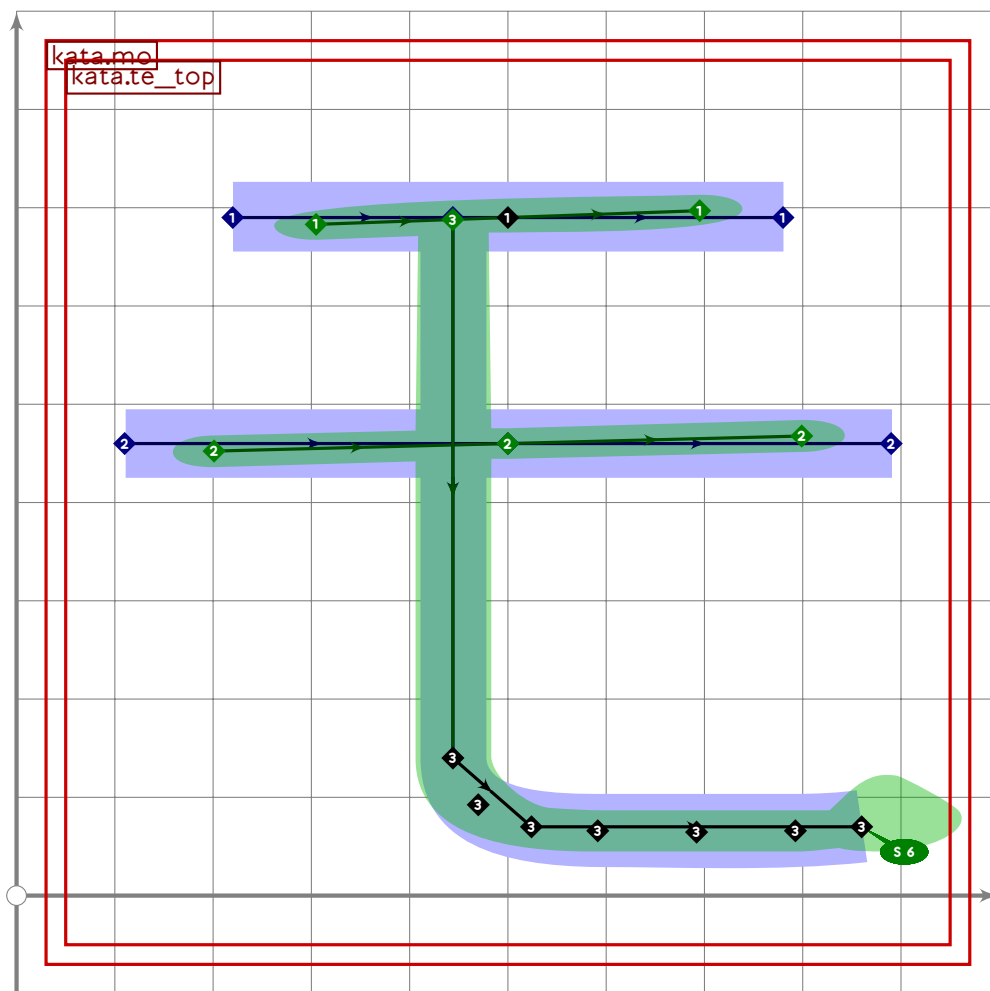


```

572
573 vardef kata.me =
574   push_pbox__toexpand("kata.me");
575
576   kata.no_stroke((720,730),(160,10));
577   set_boserif(0,0,8);
578
579   push_stroke((340,470)..(570,350)..(780,180),
580     (1.3,1.3)–(1.6,1.6)–(1.8,1.8));
581   expand_pbox;
582 enddef;

```

U+30E2  
tsuku.uni30E2



```

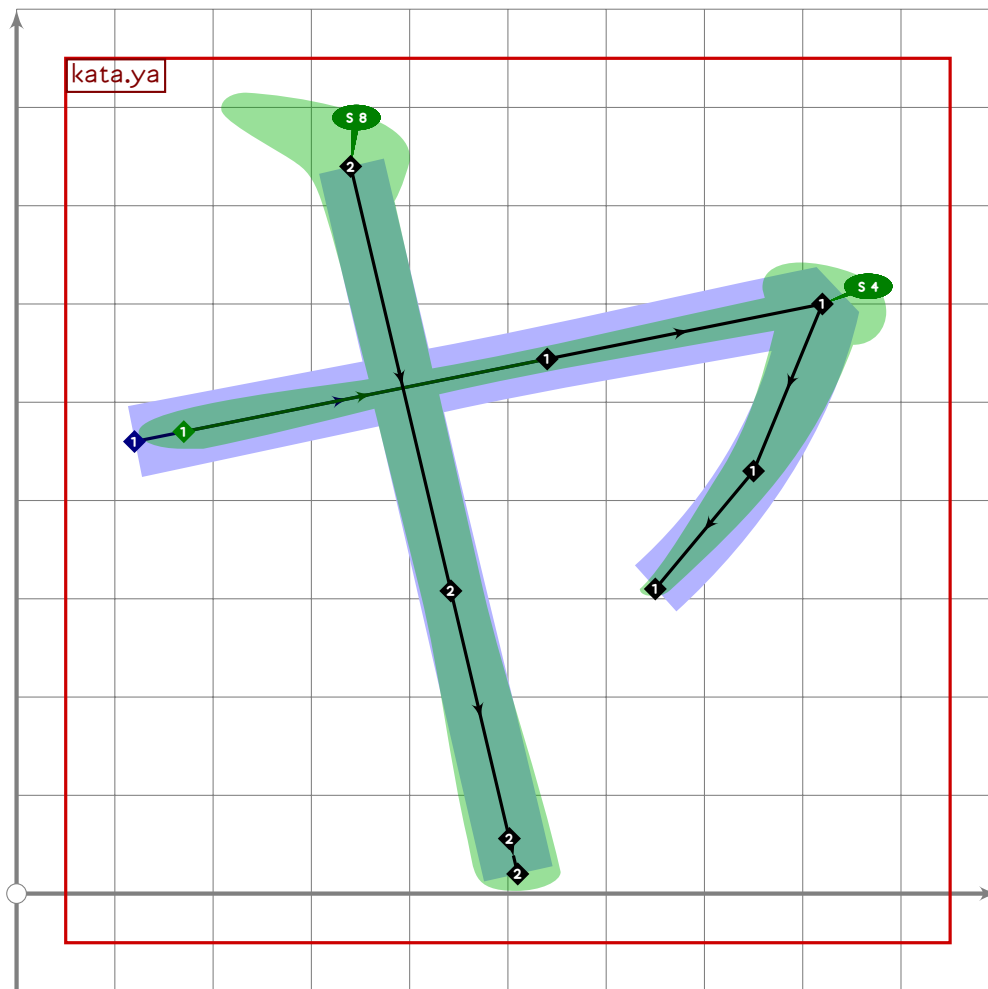
583
584 vardef kata.mo =
585   push_pbox_toexpand("kata.mo");
586
587   kata.te_top;
588
589   push_stroke((point 0.8 of get_stroke(-1))-
590     (xpart point 0.8 of get_stroke(-1),140){dir 274}..
591     (80+xpart point 0.8 of get_stroke(-1),70)..tension 2.1..(860,70),
592     (1.5,1.5)-(1.6,1.6)-(2,2)-(1.9,1.9));
593   set_boserif(0,3,6);
594   expand_pbox;
595 enddef;
596

```

**KATA**

Katakana Yayuyo

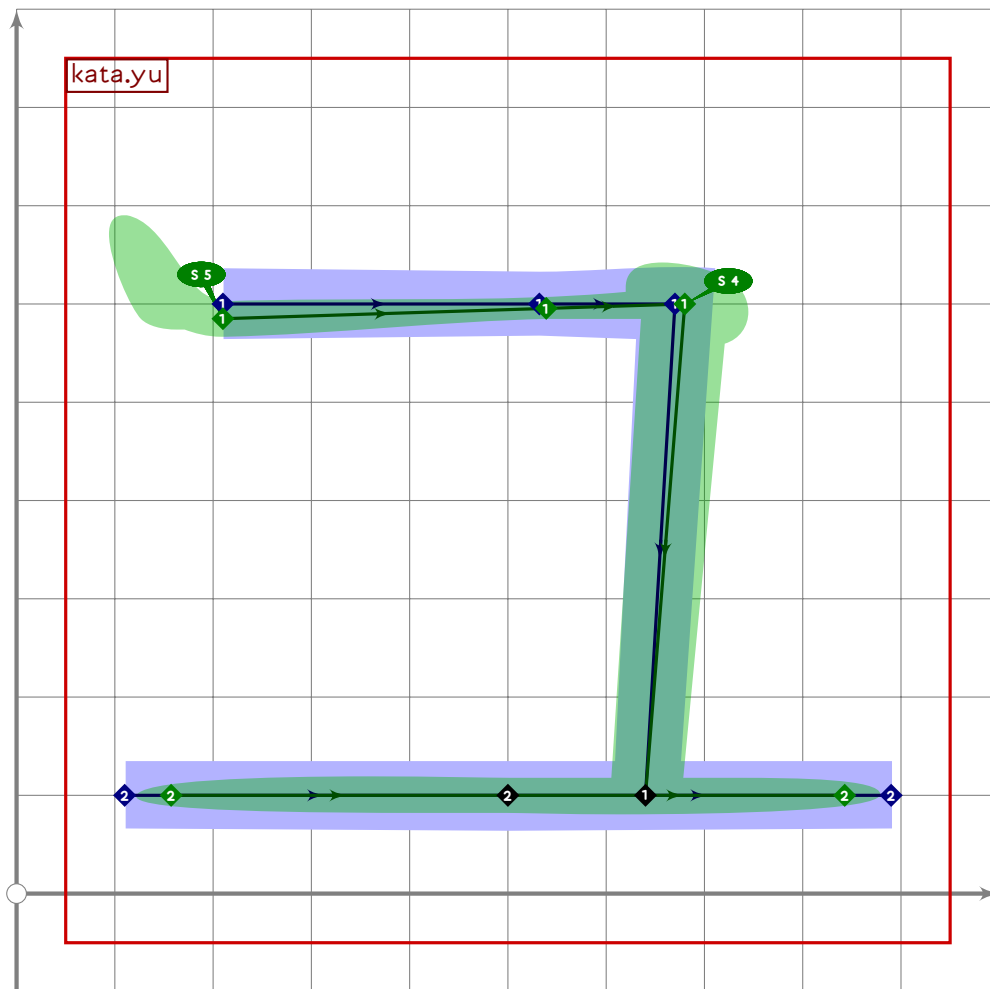
597 %%%%%%%%% KATAKANA YAYUYO



```

598
599 vardef kata.ya =
600   push_pbox_toexpand("kata.ya");
601
602   push_stroke((120,460)-(820,600)..(750,430)..(650,310),
603     (0.77,2.9)-(1.3,1.3)-(1.7,1.7)-(1.4,1.4)-(1,1));
604   replace_strokep(0)(insert_nodes(oldp)(0.6));
605   set_botip(0,2,0);
606   set_boserif(0,0,5);
607   set_boserif(0,2,4);
608
609   push_stroke((340,740)-(510,20),
610     (1.5,1.5)-(1.4,1.4)-(1.7,1.7)-(1.7,1.7));
611   replace_strokep(0)(insert_nodes(oldp)(0.6,0.95));
612   set_boserif(0,0,8);
613   expand_pbox;
614 enddef;

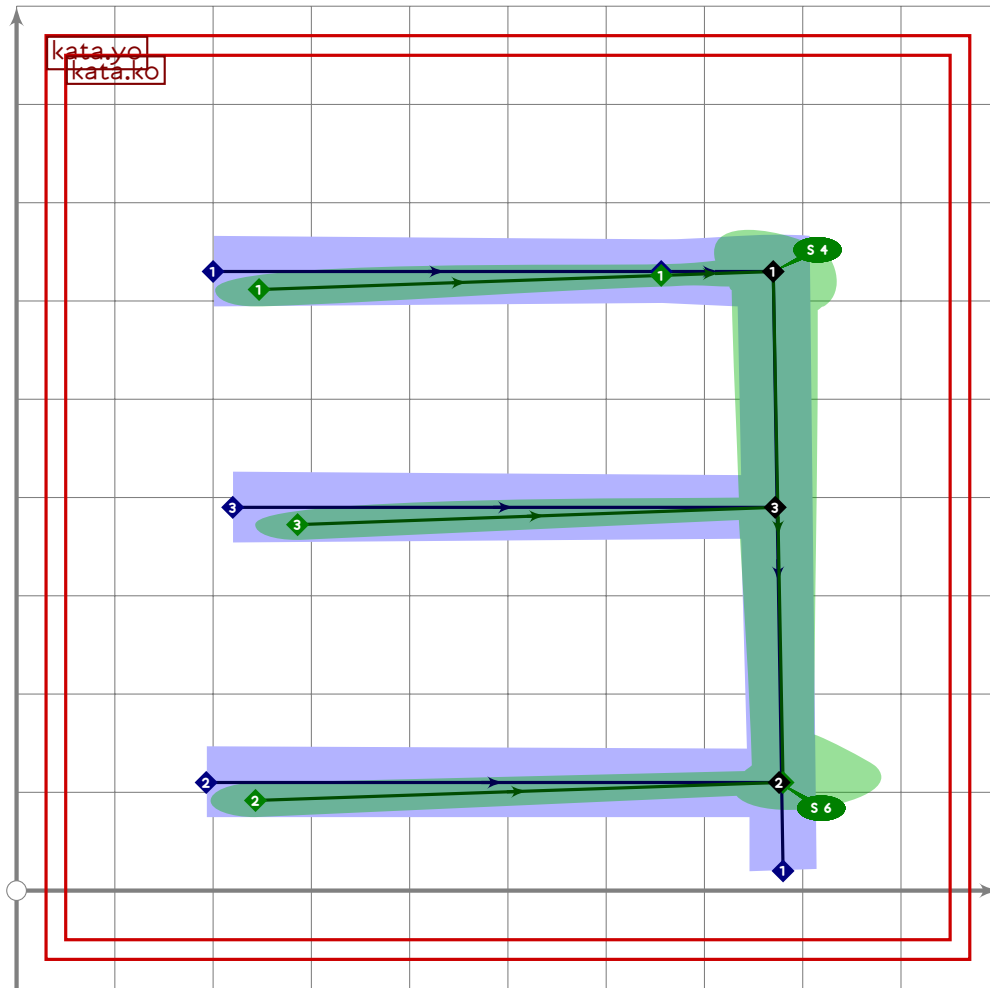
```



```

615
616 vardef kata.yu =
617   push_pbox_toexpand("kata.yu");
618
619   push_stroke((210,600)-(15*mincho)-(670+10*mincho,600)-(640,100),
620     (1.8,1.8)-(1.2,1.2)-(1.7,1.7)-(1.5,1.5));
621   replace_stroke(0)(insert_nodes(oldp)(0.7));
622   set_botip(0,2,1);
623   set_boserif(0,0,5);
624   set_boserif(0,2,4);
625
626   push_stroke((110,100)-(500,100)-(890,100),
627     (0.7,2.2)-(1.8,1.8)-(0.7,2.2));
628   set_boserif(0,0,5);
629   set_boserif(0,2,6);
630   expand_pbox;
631 enddef;

```



```

632
633 vardef kata.yo =
634   push_pbox_toexpand("kata.yo");
635
636   kata.ko;
637
638   push_stroke((220,390-20*mincho)-(772,390),
639     (0.77,2.7)-(1.3,1.3));
640   set_boserif(0,0,5);
641   expand_pbox;
642 enddef;
643

```

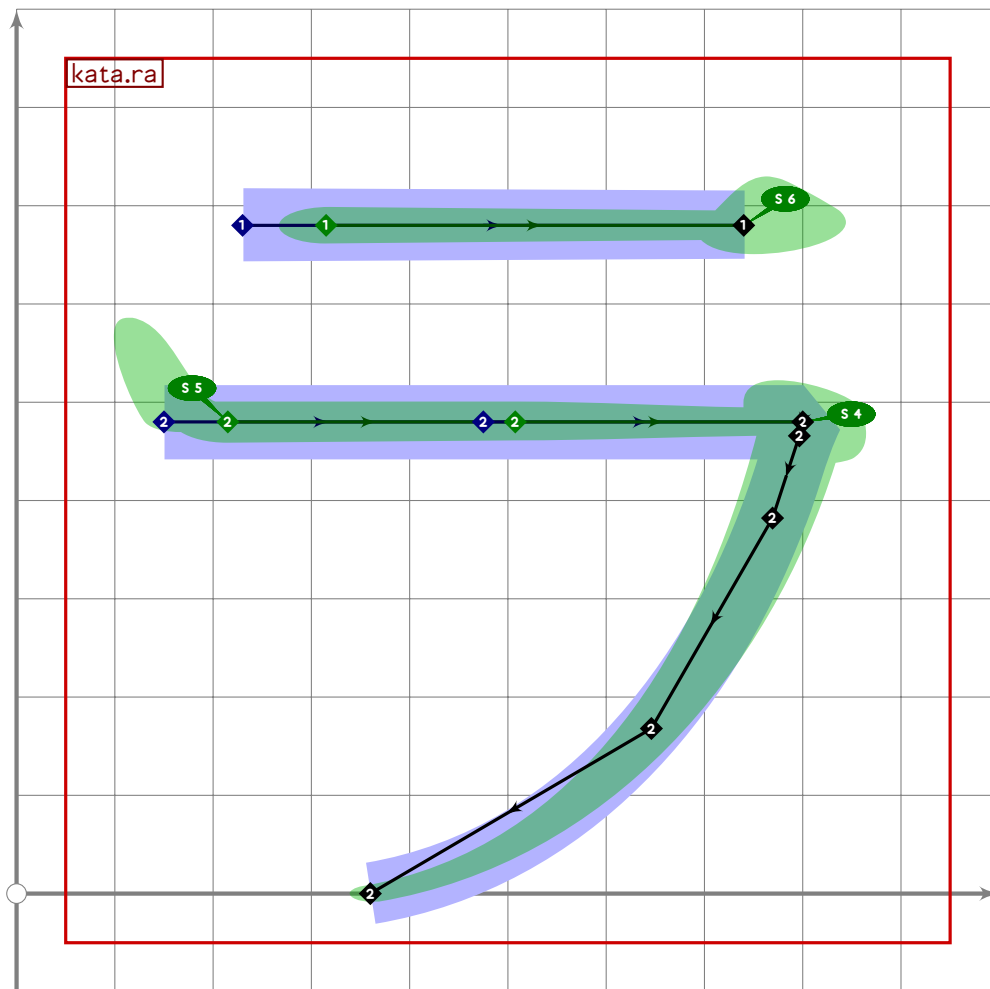
KATA

## Katakana Rarirurero

```

644 %%%%%%%%% KATAKANA RARIRURERO

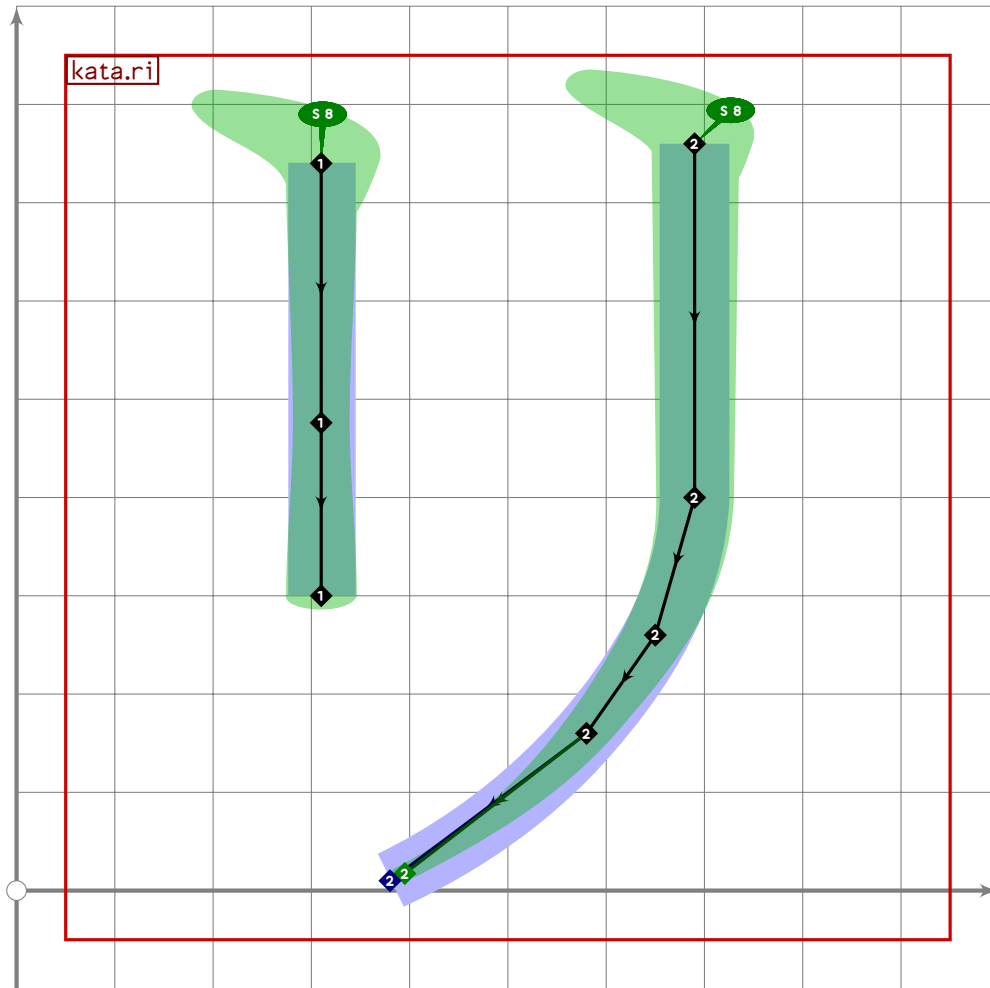
```



```

645
646 vardef kata.ra =
647   push_pbox_toexpand("kata.ra");
648
649   push_stroke((230,680)-(740,680),
650     (0.68,3.12)-(1.6,1.6));
651   set_boserif(0,0,5);
652   set_boserif(0,1,6);
653
654   kata.fu_stroke((150,480),(800,480),(360,0));
655   expand_pbox;
656 enddef;

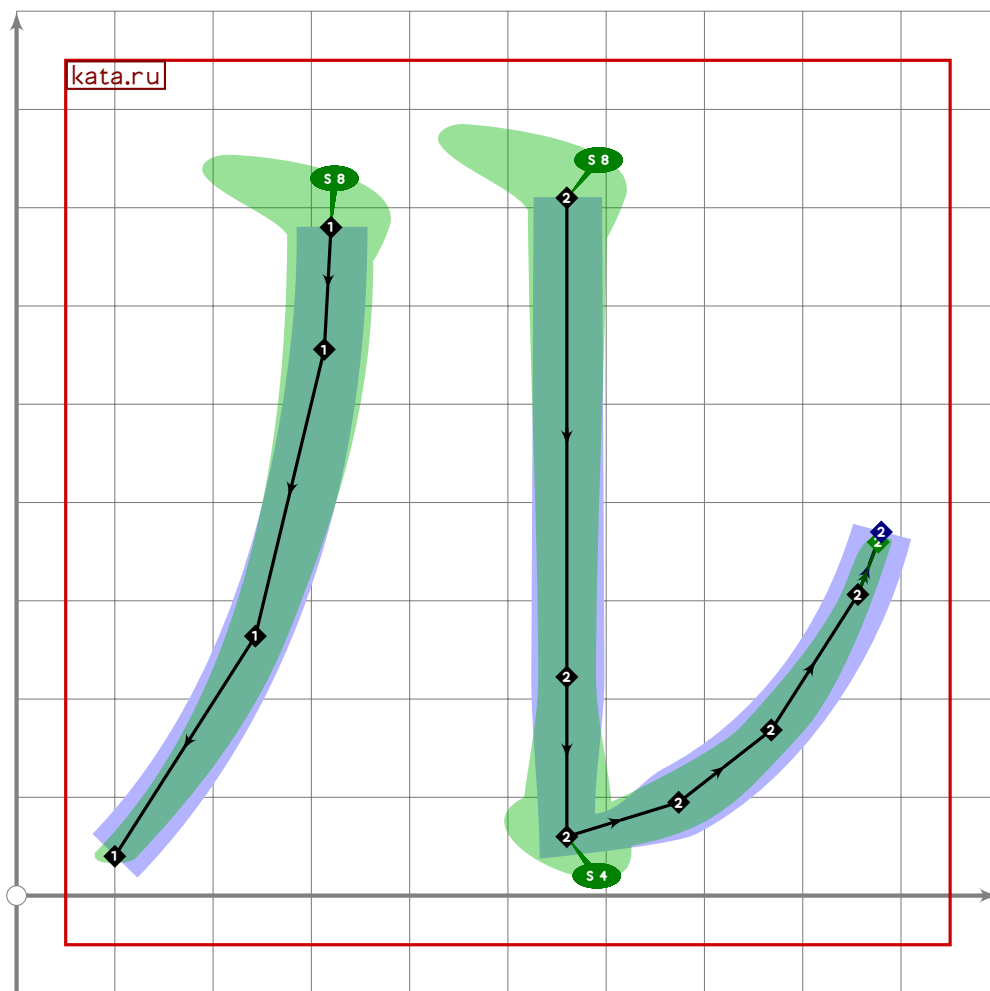
```



```

657
658 vardef kata.ri =
659   push_pbox_toexpand("kata.ri");
660
661   push_stroke((310,740)–(310,300),
662     (1.5,1.5)–(1.3,1.3)–(1.5,1.5));
663   replace_stroke(0)(insert_nodes(oldp)(0.6));
664   set_boserif(0,0,8);
665
666   push_stroke((690,760)–(690,400){dir 267}..(650,260)..(580,160)..(380,10),
667     (1.7,1.7)–(1.6,1.6)–(1.5,1.5)–(1.3,1.3)–(0.8,1));
668   set_boserif(0,0,8);
669   expand_pbox;
670 enddef;

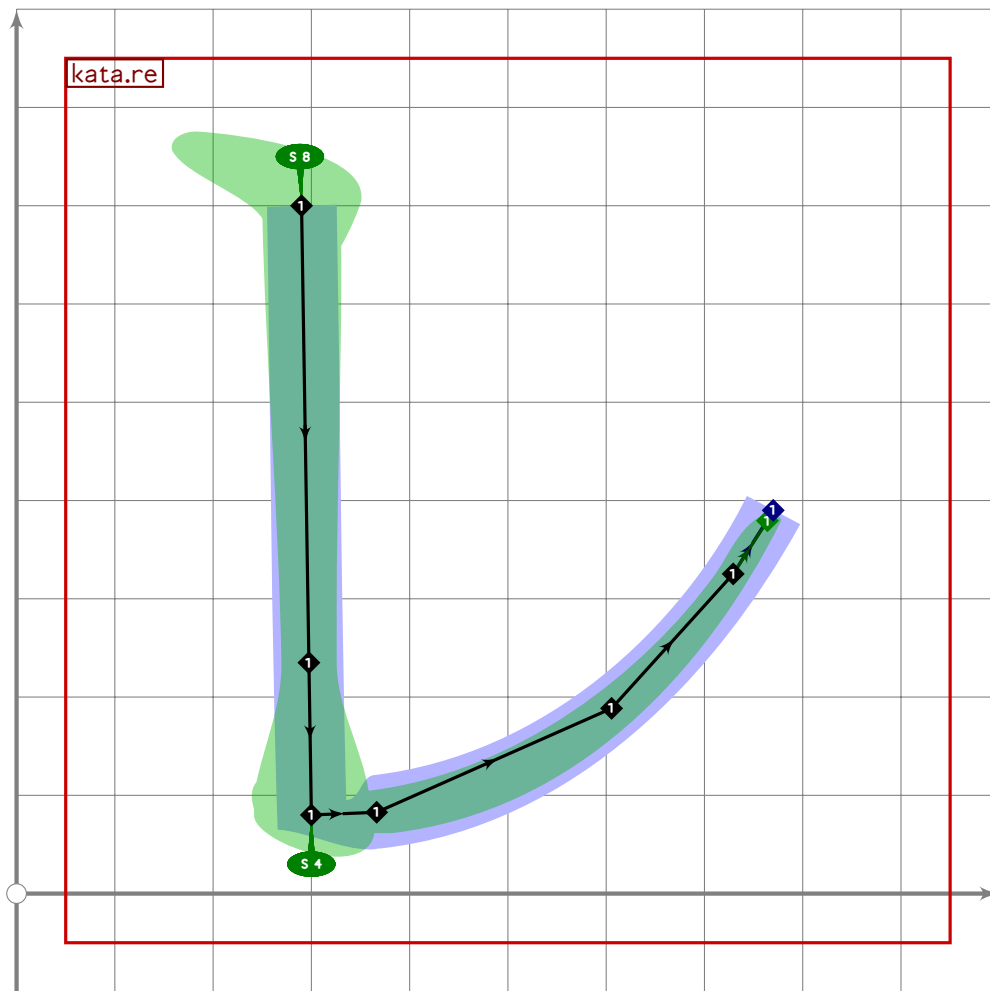
```



```

671
672 vardef kata.ru =
673   push_pbox_toexpand("kata.ru");
674
675   kata.no_stroke((320,680),(100,40));
676   set_boserif(0,0,8);
677
678   kata.no_stroke((880,370),(560,60));
679   replace_strokep(0)(insert_nodes((560,710)-reverse oldp)(0.75,1.5));
680   replace_strokeq(0)((1.6,1.6)-(1.3,1.3)-(1.8,1.8)-
681     (2,2)-(1.4,1.4)-(1,1)-(0.8,1));
682   set_botip(0,1,0);
683   set_boserif(0,0,8);
684   set_boserif(0,2,4);
685   expand_pbox;
686 enddef;

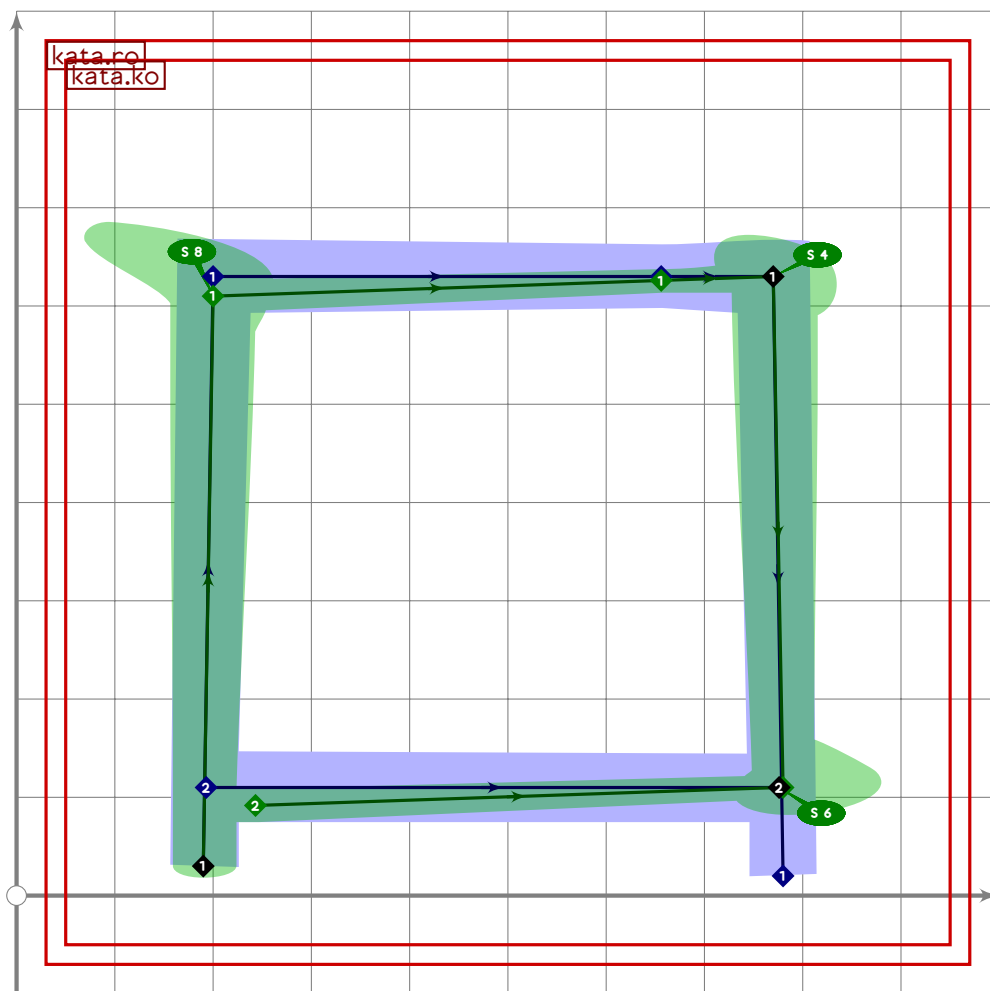
```



```

687
688 vardef kata.re =
689   push_pbox__toexpand("kata.re");
690
691   kata.no__stroke((770,390),(300,80));
692   replace__strokep(0)(insert__nodes((290,700)-reverse oldp)(0.75,1.2));
693   replace__strokeq(0)((1.6,1.6)-(1.3,1.3)-(2.1,2.1)-
694     (2,2)-(1.4,1.4)-(1,1)-(0.8,1));
695   set__botip(0,1,0);
696   set__boserif(0,0,8);
697   set__boserif(0,2,4);
698   expand__pbox;
699 enddef;

```



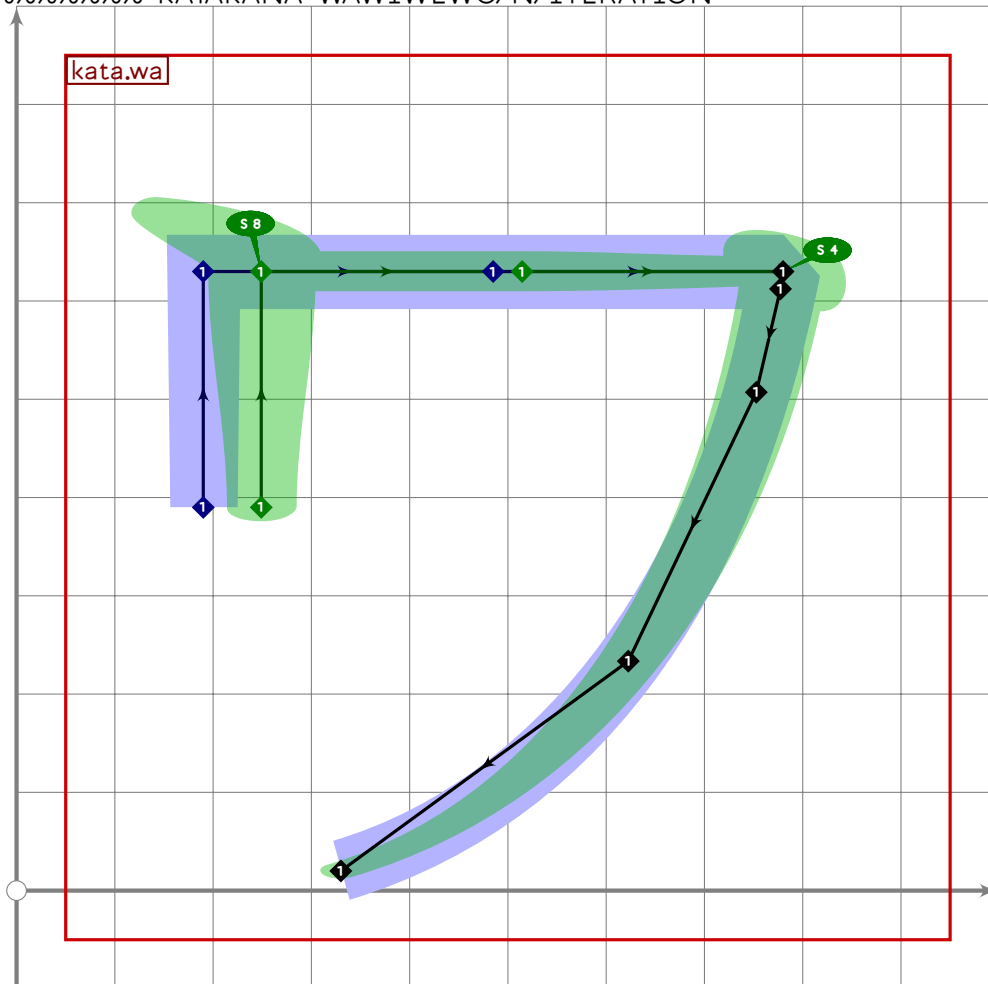
```

700
701 vardef kata.ro =
702   push_pbox_toexpand("kata.ro");
703
704   kata.ko;
705
706   replace_strokep(-1)((190,30)-oldp);
707   replace_strokeq(-1)((14,14)-(1.7,1.7)-(subpath (1,infinity) of oldq));
708   set_botip(-1,1,1);
709   set_botip(-1,2,whatever);
710   set_botip(-1,3,1);
711   set_boserif(-1,0,whatever);
712   set_boserif(-1,1,8);
713   set_boserif(-1,2,whatever);
714   set_boserif(-1,3,4);
715   set_boserif(0,0,whatever);
716   expand_pbox;
717 enddef;
718

```

# Katakana Wawiwewo/N/Iteration

719 %%%%%%%%% KATAKANA WAWIWEWO/N/ITERATION



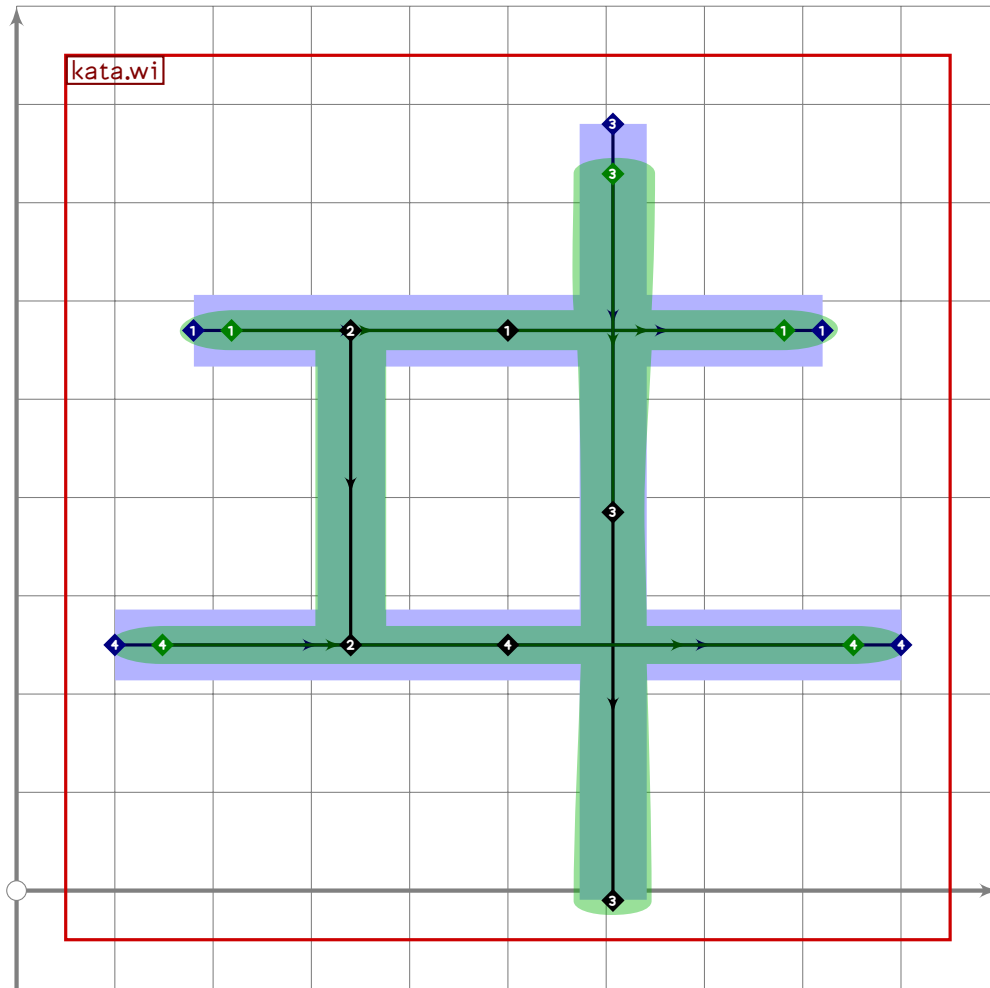
```

720
721 vardef kata.wa =
722   push_pbox_toexpand("kata.wa");
723
724   kata.fu_stroke((190,630),(780,630),(330,20));
725   replace_strokep(0)((xpart point 0 of oldp,390)-oldp);
726   replace_strokeq(0)((1.5,1.5)-oldq);
727   set_botip(0,1,1);
728   set_botip(0,3,0);
729   set_boserif(0,0,whatever);
730   set_boserif(0,1,8);
731   set_boserif(0,2,whatever);
732   set_boserif(0,3,4);
733   expand_pbox;
734 enddef;

```

**KATA**

U+30F0  
tsuku.uni30F0



```

735
736 vardef kata.wi =
737   push_pbox_toexpand("kata.wi");
738
739   x1=100;
740   x2=180;
741   x3=0.25[x2,x5];
742   x4=0.667[x2,x5];
743   (x5+x2)/2=(x1+x6)/2=500;
744   y1=-10;
745   y2=250;
746   y3=570;
747   y4=780;
748   push_stroke((x2,y3)-(x5,y3),
749     (0.7,3.3)-(1.8,1.8)-(0.7,3.3));
750   replace_strokep(0)(insert_nodes(oldp)(0.5));
751   set_boserif(0,0,5);
752   set_boserif(0,2,6);
753
754   push_stroke((x3,y3)-(x3,y2),
755     (1.5,1.5)-(1.5,1.5));

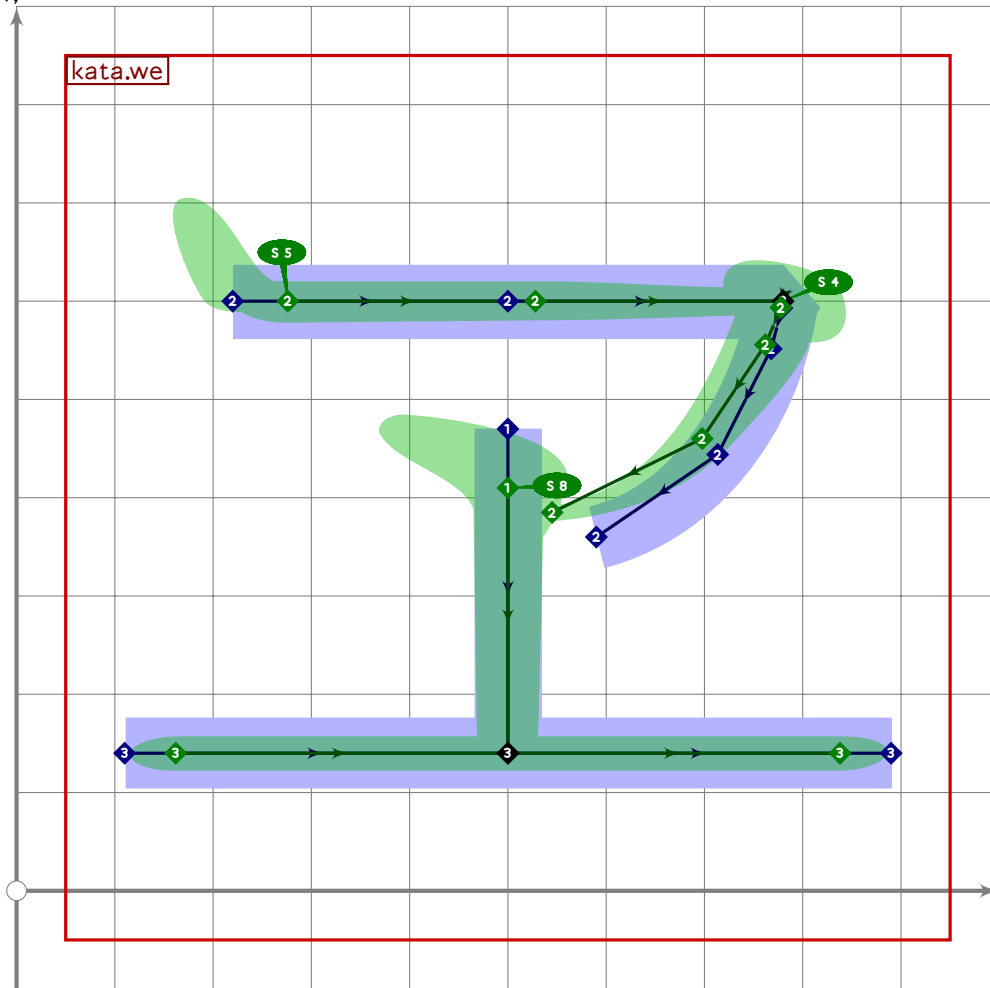
```

KATA

```

756
757 push_stroke((x4,y4)-(x4,0.5[y4,y1])-(x4,y1),
758   (0.75,2.65)-(1.4,1.4)-(1.6,1.6));
759 set_boserif(0,0,8);
760
761 push_stroke((x1,y2)-(x6,y2),
762   (0.7,3.3)-(1.8,1.8)-(0.7,3.3));
763 replace_strokep(0)(insert_nodes(oldp)(0.5));
764 set_boserif(0,0,5);
765 set_boserif(0,2,6);
766 expand_pbox;
767 enddef;

```



```

768
769 vardef kata.we =
770   push_pbox_toexpand("kata.we");
771
772   push_stroke((500,470-60*mincho)-(500,140),
773     (1.5,1.5)-(1.4,1.4));
774   set_boserif(0,0,8);
775
776   kata.fu_stroke((220,600),(780,600),

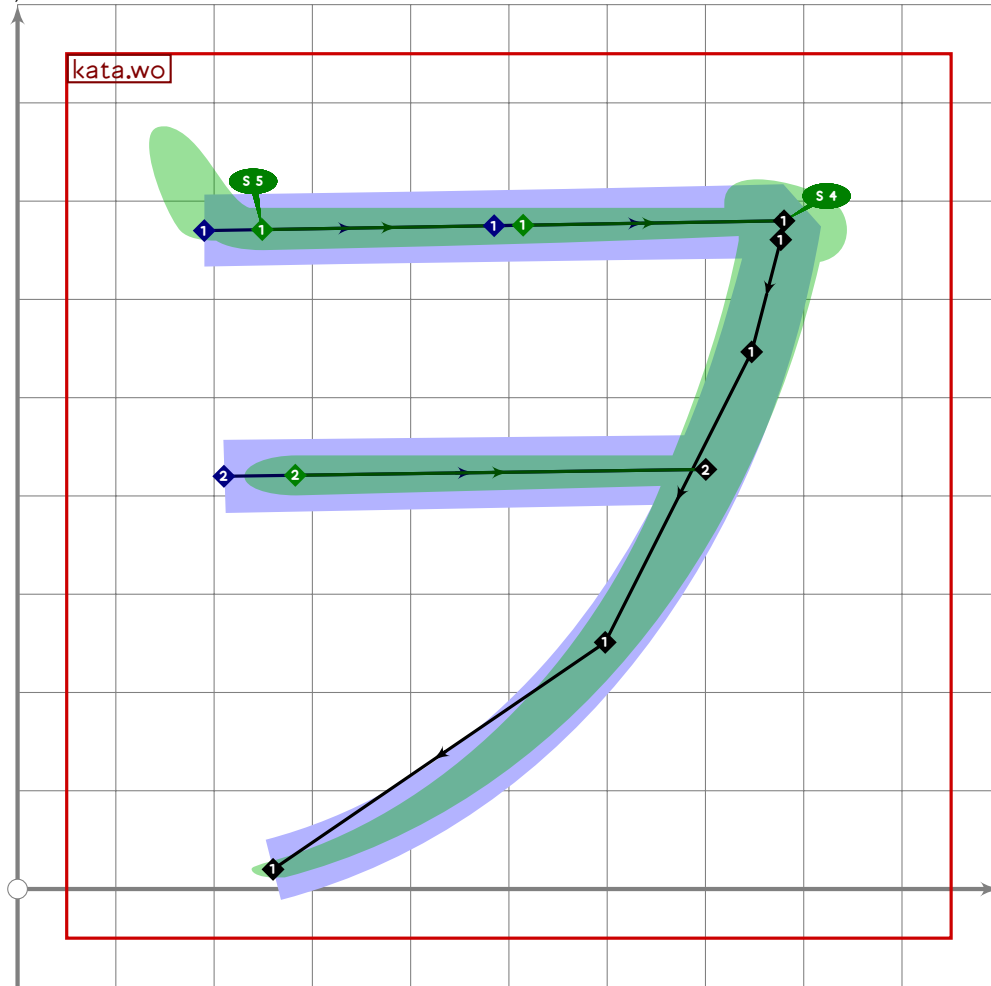
```

U+30F2  
tsuku.uni30F2

```

777 (0.5*mincho)[(590,360),point 0 of get_stroke(0)];
778
779 push_stroke((110,140)–(500,140)–(890,140),
780 (0.7,2.9)–(1.7,1.7)–(0.7,2.9));
781 set_boserif(0,0,5);
782 set_boserif(0,2,6);
783 expand_pbox;
784 enddef;

```

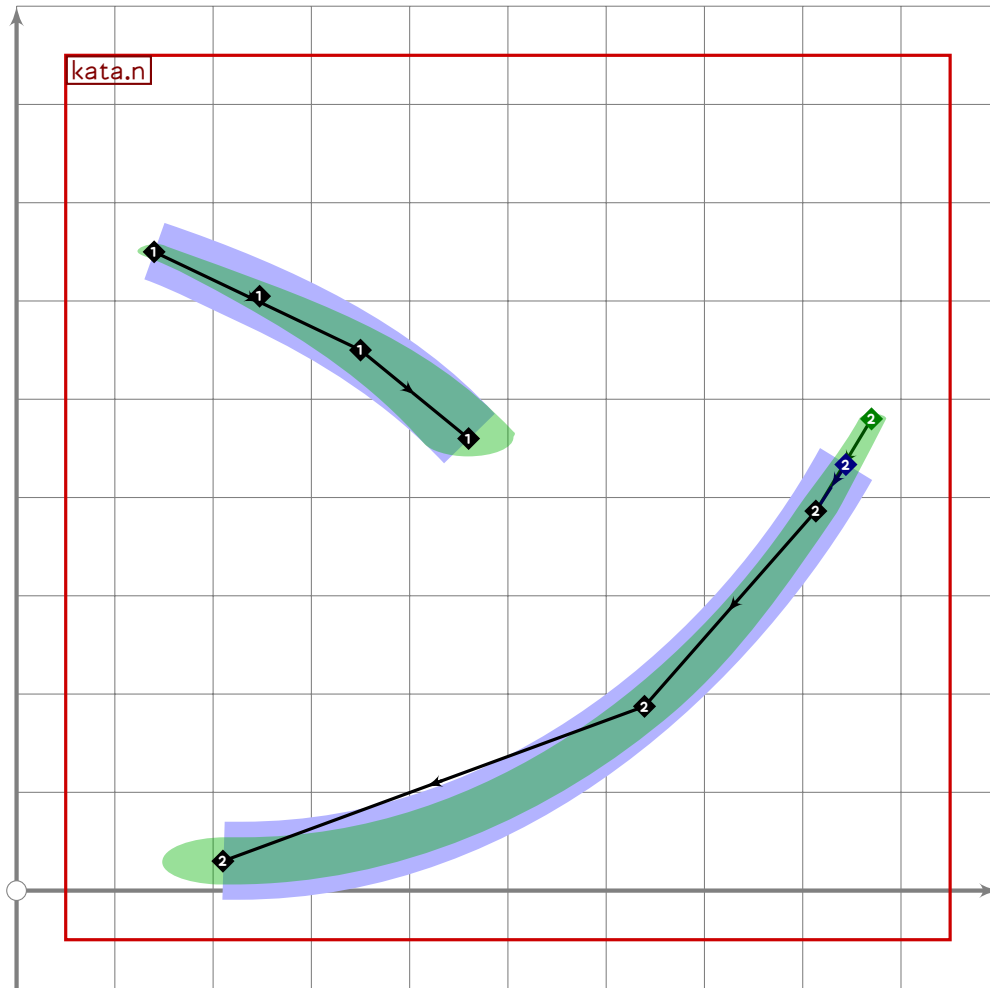


```

785
786 vardef kata.wo =
787   push_pbox_toexpand("kata.wo");
788
789   kata.fu_stroke((190,670),(780,680),(260,20));
790
791   z1=get_stroke(0) intersectionpoint ((0,420)–(1000,430));
792   push_stroke((210,420)–z1,
793 (0.7,3.3)–(1.6,1.6));
794   set_boserif(0,0,5);
795   expand_pbox;
796 enddef;

```

**KATA**

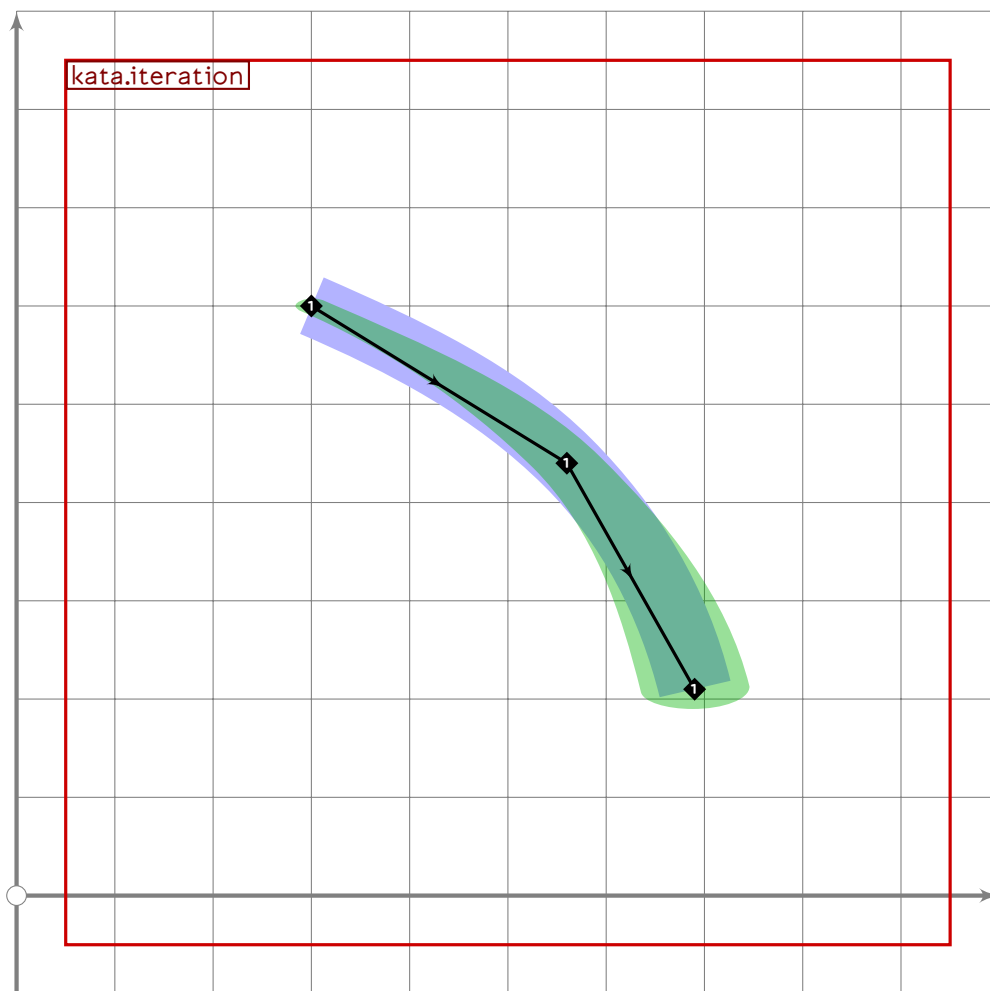


```

797
798 vardef kata.n =
799   push_pbox__toexpand("kata.n");
800
801   push_stroke((140,650)..tension 1.2..(350,550)..(460,460),
802     (1,1)..(1.6,1.6)..(1.8,1.8));
803
804   kata.no_stroke((870,480),(210,30));
805   replace_strokeq(0)((0.9,0.9)-(1.1,1.1)-(1.4,1.4)-(2.2,2.2));
806   set_boserif(0.4,5);
807   expand_pbox;
808 enddef;

```

U+30FD  
tsuku.uni30FD



```

809
810 vardef kata.iteration =
811   push_pbox__toexpand("kata.iteration");
812
813   push_stroke((300,600){curl 0.2}..(560,440)..(690,210),
814     (1,1)-(1.5,1.5)-(2,2));
815   expand_pbox;
816 enddef;

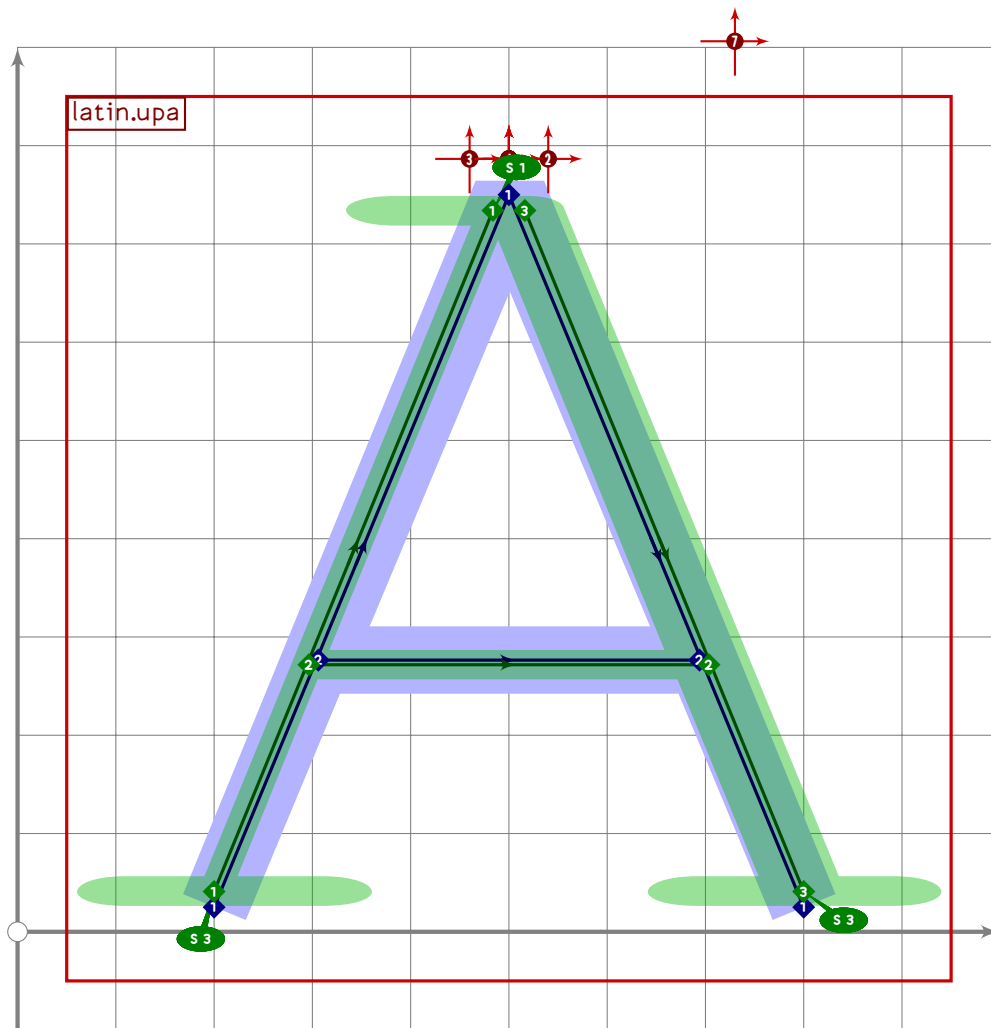
```

# latin.mp

```

1 %
2 % Latin and related letters for Tsukurimashou
3 % Copyright (C) 2011, 2012, 2013 Matthew Skala
4 %
5-29 [Standard copyright notice]
30
31 inclusion_lock(latin);
32
33

```



```

34
35 vardef latin.upa =
36   push_pbox_toexpand("latin.upa");
37   z1=(200,latin_wide_low_v);
38   z2=(500,latin_wide_high_v);
39   z3=(800,latin_wide_low_v);
40
41   if do_alteration:
42     z4=whatever[z1,z2+alternate_adjust*left/2]+(2,0);

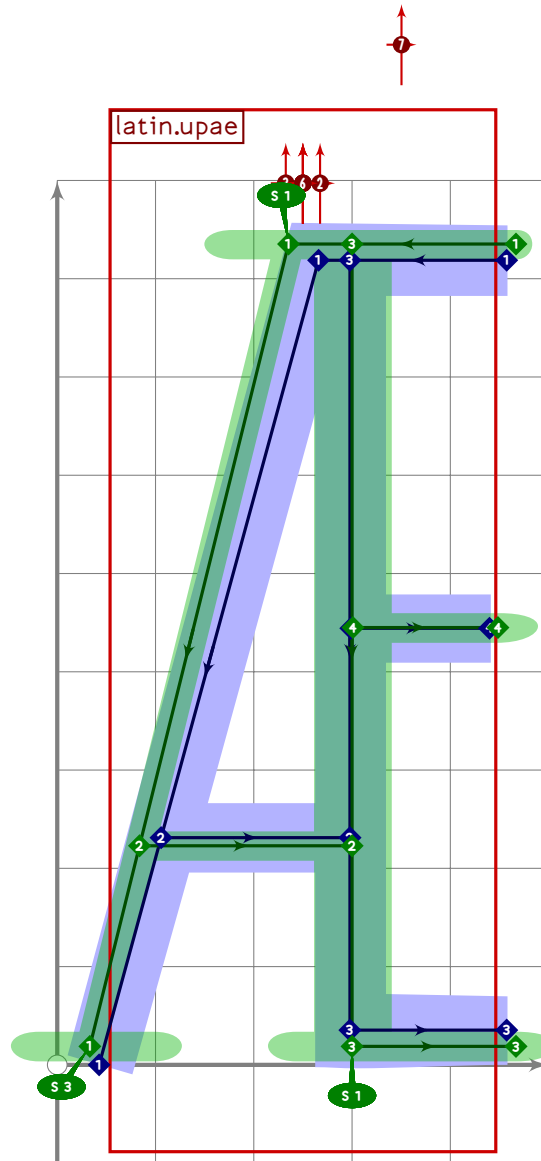
```

LATI

```

43  z5=whatever[(z2+alternate_adjust*right/2),z3]-(2,0);
44  y4=y5=vmetric(0.333);
45
46  push__stroke(z1-(z2+alternate_adjust*left/2),(1.6,1.6)-(1.6,1.6));
47  set__boalternate(0);
48  set__botip(0,1,0);
49  set__boserif(0,0,3);
50  set__boserif(0,1,1);
51
52  push__stroke(z4-z5,(1.6,1.6)-(1.6,1.6));
53  set__boalternate(0);
54
55  push__stroke((z2+alternate_adjust*right/2)-z3,(1.6,1.6)-(1.6,1.6));
56  set__boserif(0,1,3);
57  else:
58  z4=whatever[z1,z2]+(2,0);
59  z5=whatever[z2,z3]-(2,0);
60  y4=y5=vmetric(0.333);
61
62  push__stroke(z1-z2-z3,(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
63  set__botip(0,1,0);
64  set__boserif(0,0,3);
65  set__boserif(0,1,1);
66  set__boserif(0,2,3);
67
68  push__stroke(z4-z5,(1.6,1.6)-(1.6,1.6));
69  fi;
70
71  tsu__accent.shift__anchors(ypart olda>vmetric(0.52))
72  (((0,0) transformed tsu__xf.cap__upper__accent)-
73  ((0,0) transformed accent__default[anc__upper]));
74  expand__pbox;
75 endif;

```



```

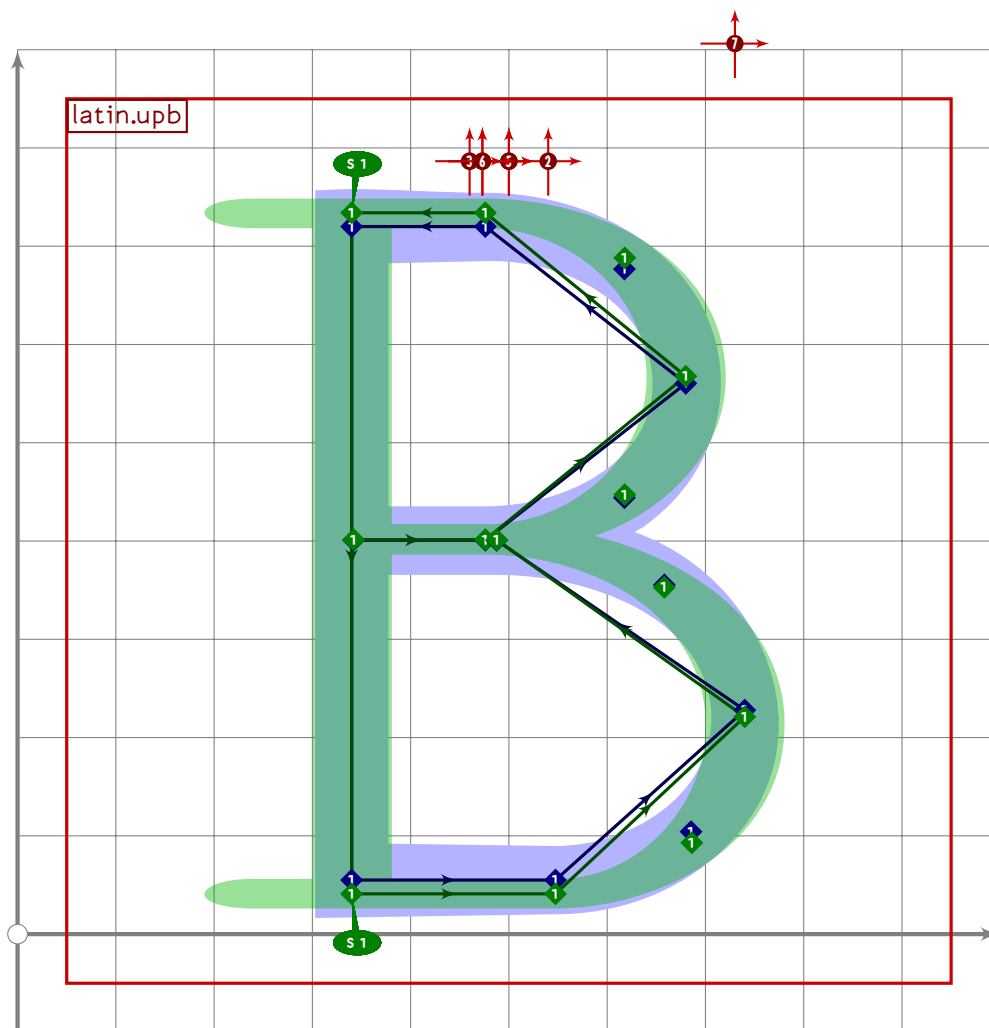
76
77 vardef latin.upae =
78   push_pbox_toexpand("latin.upae");
79   y1=y2=latin_wide_high_h;
80   y3=y4=latin_wide_low_h;
81   y5=y6=vmetric(0.522);
82
83   (x1+x7)/2=500;
84   x2=x3;
85   x1=x4;
86   x5=x2+2;
87   x6=0.89[x2,x1];
88   (x1-x2)=(y2-y3)*0.55;
89
90   y7=latin_wide_low_v;
91   x7=(-1.6)[x2,x1];

```

```

92  z10=(-0.2)[z2,z1]+2.2*alternate_adjust*left;
93  z8=whatever[z7,z10];
94  z9=whatever[z2,z3];
95  y8=y9=vmetric(0.250);
96
97  push__stroke(z1-z10-z7,(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
98  set__botip(0,1,1);
99  set__boserif(0,1,1);
100 set__boserif(0,2,3);
101 set__boalternate(0);
102
103 push__stroke(z8-z9,(1.6,1.6)-(1.6,1.6));
104 set__boalternate(0);
105
106 push__stroke(z2-z3-z4,(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
107 set__botip(0,1,1);
108 set__boserif(0,1,1);
109
110 push__stroke(z5-z6,(1.6,1.6)-(1.6,1.6));
111
112 tsu_accent.shift__anchors(ypart olda>vmetric(0.52))
113   (((0,0) transformed tsu_xf.cap_upper_accent)-
114    ((0,0) transformed accent_default[anc_upper]));
115 expand__pbox;
116 endif;

```



```

117
118 vardef latin.upb =
119   push_pbox_toexpand("latin.upb");
120   latin.upp_base(340);
121
122   x6=x5;
123   x7=0.61[x5,x3];
124   x8=x5+400;
125
126   y6=y7=latin_wide_low_h;
127   y8=0.5[y6,y1];
128
129   z9=z2;
130
131   replace_strokep(0)(oldp-z6-z7{right}..z8..{left}z9);
132   replace_strokep(0)(subpath (0,797) of oldp);
133   replace_strokeq(0)(oldq-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
134
135   set_botip(0,4,1);
136   set_botip(0,5,1);

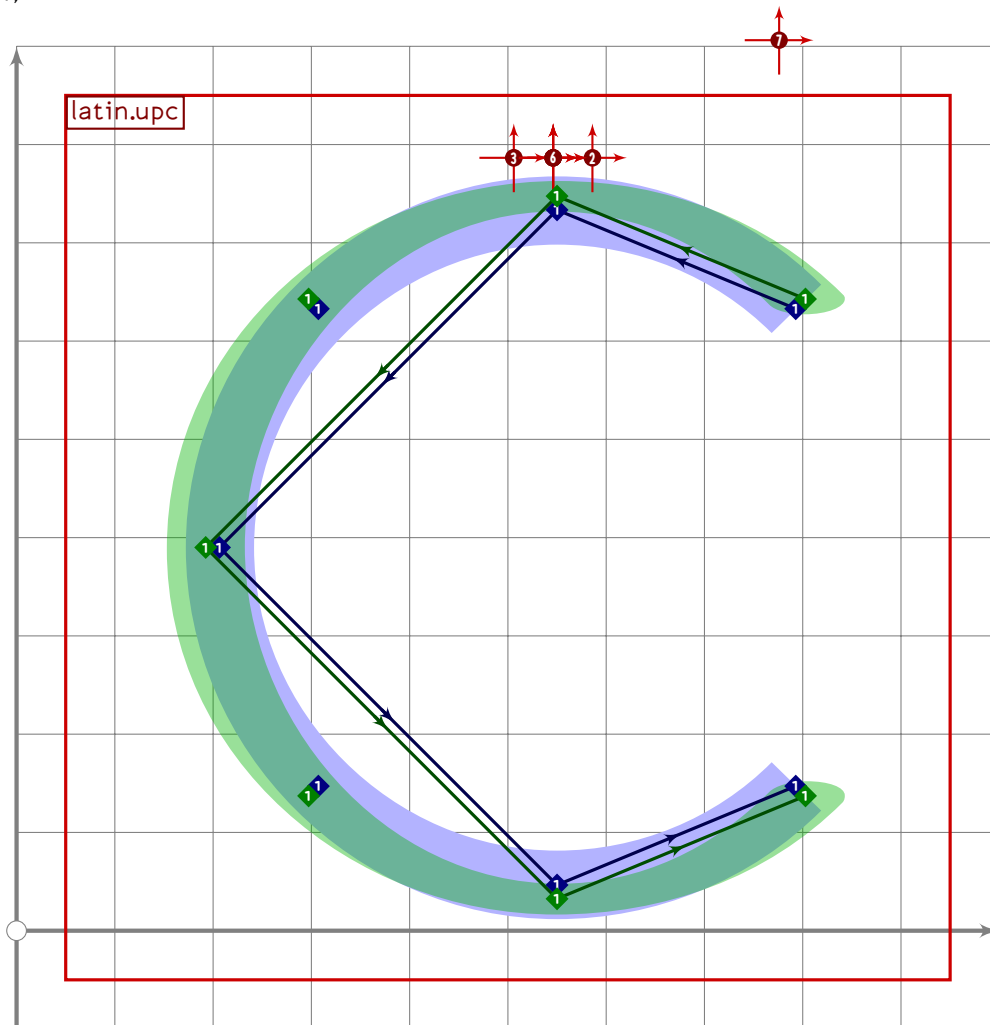
```

U+FF23  
tsuku.uniFF23

```

137 set_boserif(0,4,1);
138 set_boserif(0,5,1);
139
140 tsu_accent.shift_anchors(ypart olda>vmetric(0.52))
141   (((0,0) transformed tsu_xf.cap_upper_accent)-
142    ((0,0) transformed accent_default[anc_upper]));
143 tsu_accent.shift_anchors((ai=anc_ring) or (ai=anc_upper))((-27,0));
144 expand_pbox;
145 enddef;

```



```

146
147 vardef latin.upc =
148   push_pbox_toexpand("latin.upc");
149   push_stroke(
150     (subpath (0.5,3.5) of ((1,0)..(0,1)..(-1,0)..(0,-1)..cycle))
151     scaled ((latin_wide_high_r-latin_wide_low_r)/2)
152     shifted (centre_pt+(50,0)),
153     (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
154
155   tsu_accent.shift_anchors(ypart olda>vmetric(0.52))
156     (((0,0) transformed tsu_xf.cap_upper_accent)-

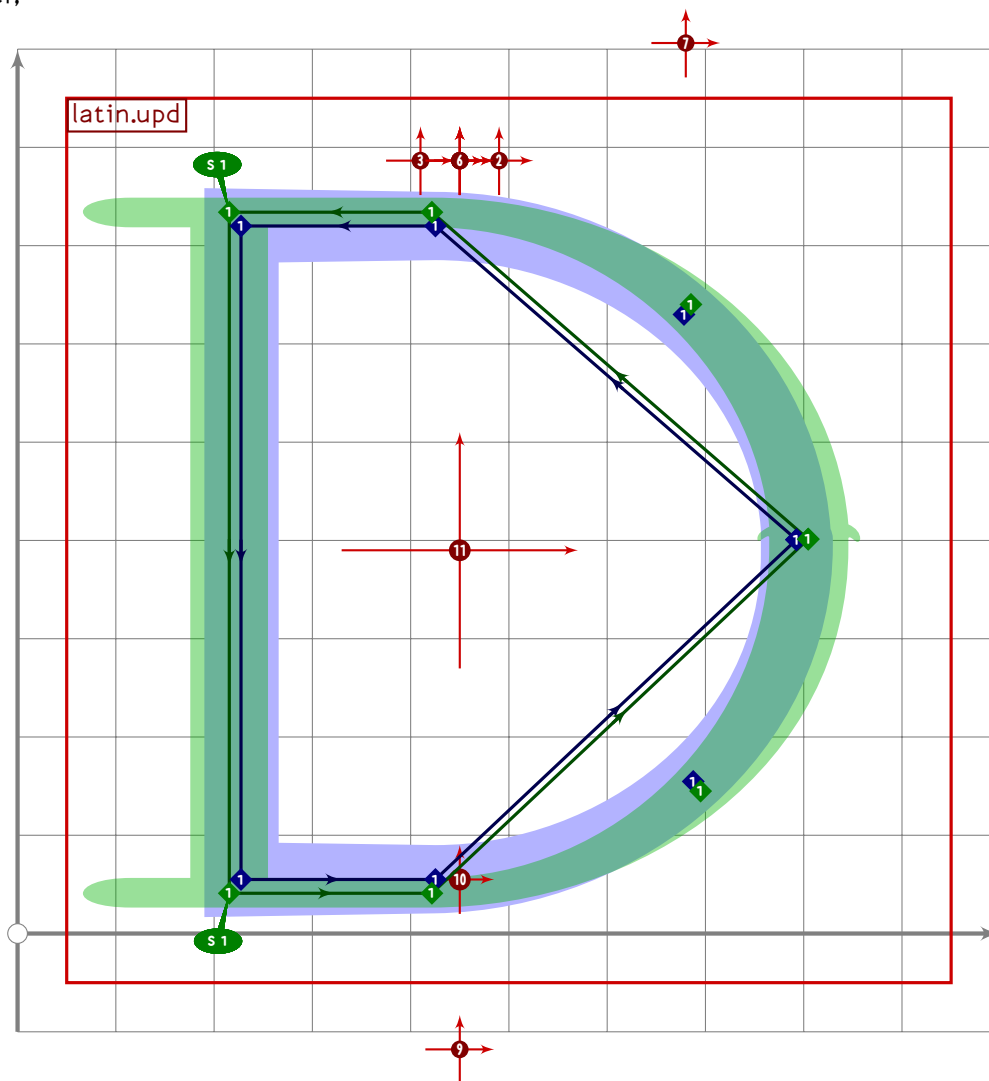
```

**LATI**

```

157 ((0,0) transformed accent_default[anc_upper])+(46,0));
158 expand_pbox;
159 enddef;

```



```

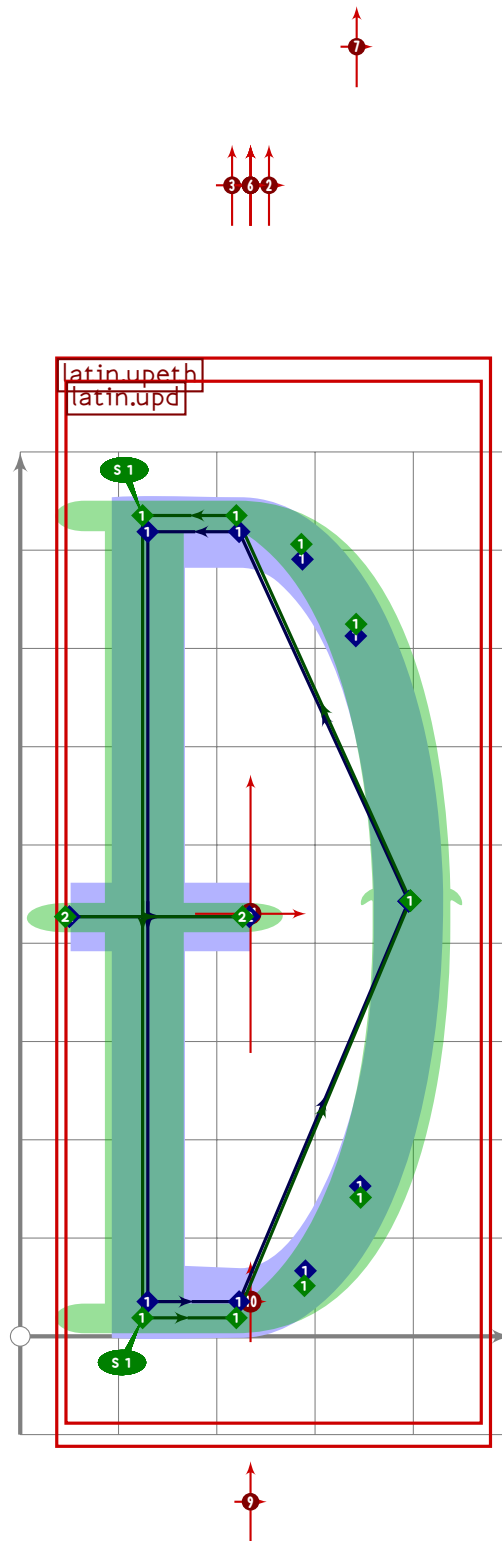
160
161 vardef latin.upd =
162   push_pbox_toexpand("latin.upd");
163   y1=y5=latin_wide_high_h;
164   y2=y3=latin_wide_low_h;
165   y4=0.52[y2,y1];
166
167   (x1+x4)/2=510;
168   (x4-x1)=0.85*(y1-y2);
169   x1=x2;
170   x3=x5=0.35[x1,x4];
171
172   push_stroke(z4.{left}z5-z1-z2-z3{right}..cycle,
173     (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-cycle);
174   set_botip(0,2,1);

```

```

175 set_botip(0,3,1);
176 set_boserif(0,2,1);
177 set_boserif(0,3,1);
178
179 tsu_accent.shift_anchors(ypart olda>vmetric(0.52))
180   (((0,0) transformed tsu_xf.cap_upper_accent)-
181    ((0,0) transformed accent_default[anc_upper]));
182 tsu_accent.shift_anchors(true)((-50,0));
183 expand_pbox;
184 endif;

```



```

185
186 vardef latin.upeth =
187   push_pbox_toexpand("latin.upeth");
188   latin.upd;
189   push_stroke((0.5[z1,z2]+(-170,0))-(0.5[z1,z2]+(220,0)),
190     (1.6,1.6)-(1.6,1.6));

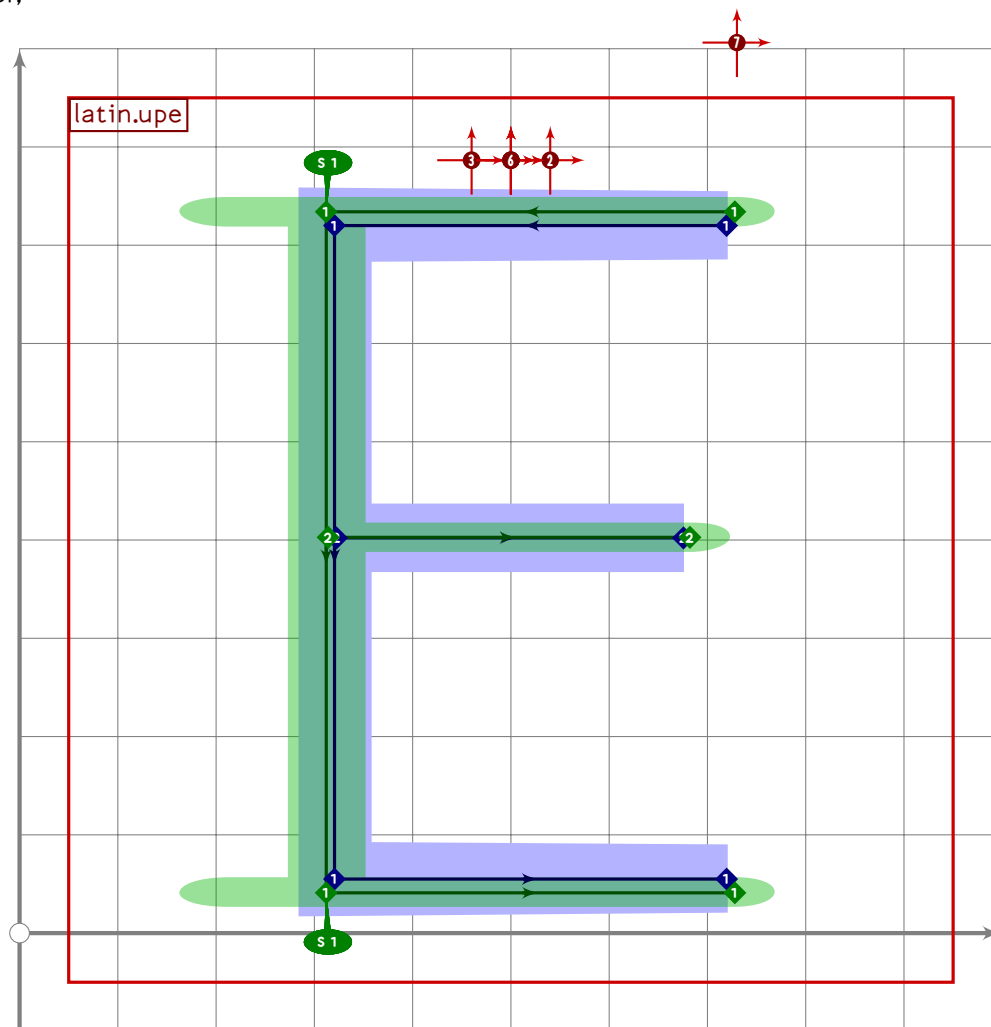
```

U+FF25  
tsuku.uniFF25

```

191
192 tsu_accent.shift_anchors(ypart olda>vmetric(0.52))
193   (((0,0) transformed tsu_xf.cap_upper_accent)-
194    ((0,0) transformed accent_default[anc_upper]));
195 expand_pbox;
196 enddef;

```



```

197
198 vardef latin.upe =
199   push_pbox_toexpand("latin.upe");
200   y1=y2=latin_wide_high_h;
201   y3=y4=latin_wide_low_h;
202   y5=y6=vmetric(0.522);
203
204   (x1+x2)/2=520;
205   x2=x3;
206   x1=x4;
207   x5=x2+2;
208   x6=0.89[x2,x1];
209   (x1-x2)=(y2-y3)*0.6;
210

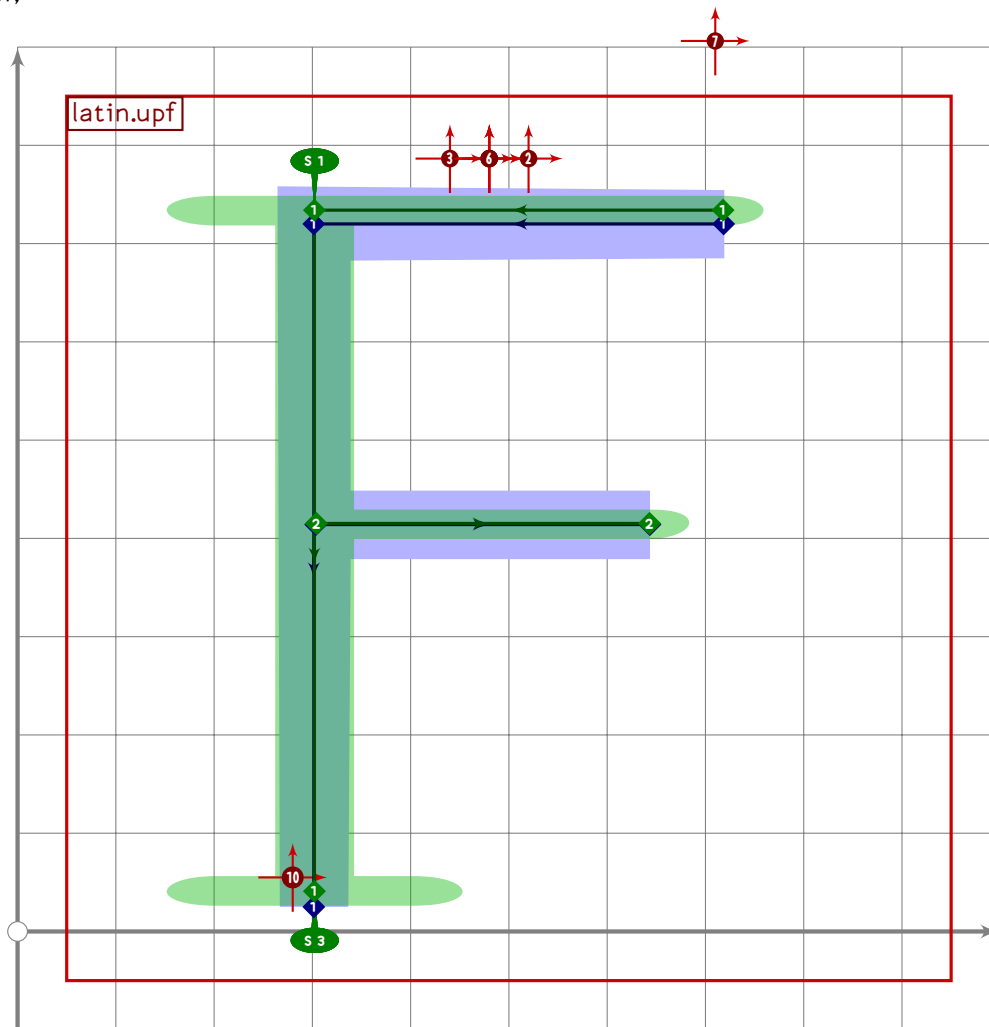
```

**LATI**

```

211 push_stroke(z1-z2-z3-z4,(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
212 set_botip(0,1,1);
213 set_botip(0,2,1);
214 set_boserif(0,1,1);
215 set_boserif(0,2,1);
216
217 push_stroke(z5-z6,(1.6,1.6)-(1.6,1.6));
218
219 tsu_accent.shift_anchors(ypart olda>vmetric(0.52))
220   (((0,0) transformed tsu_xf.cap_upper_accent)-
221    ((0,0) transformed accent_default[anc_upper]));
222 expand_pbox;
223 enddef;

```



```

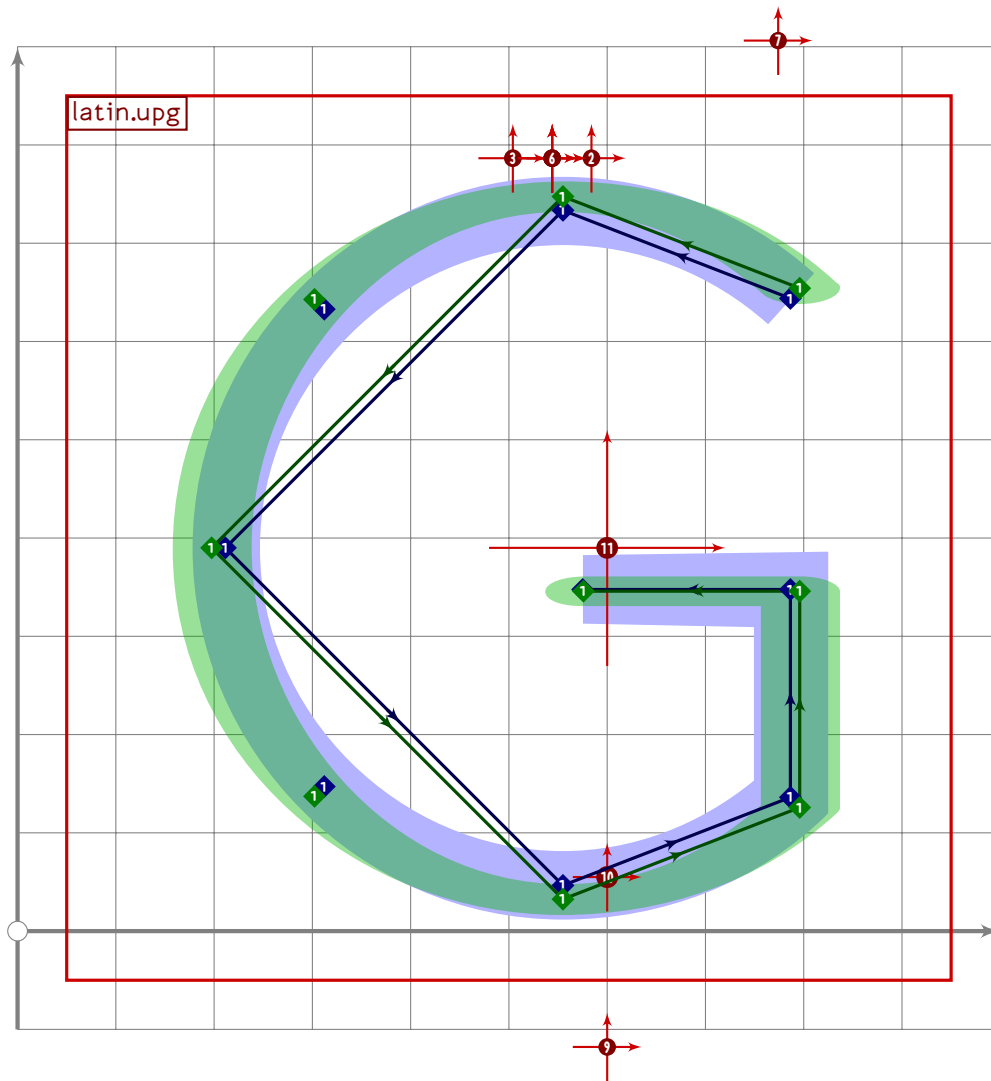
224
225 vardef latin.upf =
226   push_pbox_toexpand("latin.upf");
227   y1=y2=latin_wide_high_h;
228   y3=latin_wide_low_v;
229   y4=y5=vmetric(0.54);
230

```

```

231 (x1+x2)/2=510;
232 x3=x2;
233 x4=x2+2;
234 x5=0.82[x2,x1];
235 (x1-x2)=(y2-y3)*0.6;
236
237 push__stroke(z1-z2-z3,(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
238 set__botip(0,1,1);
239 set__boserif(0,1,1);
240 set__boserif(0,2,3);
241
242 push__stroke(z4-z5,(1.6,1.6)-(1.6,1.6));
243
244 tsu__accent.shift__anchors(ypart olda>vmetric(0.52))
245   (((0,0) transformed tsu__xf.cap_upper_accent)-
246    ((0,0) transformed accent_default[anc_upper])+(-20,0));
247 tsu__accent.shift__anchors(ai=anc_lower_connect)((-220,0));
248 expand__pbox;
249 endif;

```



```

250
251 vardef latin.upg =
252   push_pbox_toexpand("latin.upg");
253   push_stroke(
254     (subpath (0.53,3.47) of ((1,0)..(0,1)..(-1,0)..(0,-1)..cycle))
255     scaled ((latin_wide_high_r-latin_wide_low_r)/2)
256     shifted (centre_pt+(55,0)),
257     (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-
258     (1.6,1.6)-(1.6,1.6));
259
260   x1=xpart point 4 of get_stroke(0);
261   y1=y2=vmetric(0.44);
262   x1-x2=y1-(ypart point 4 of get_stroke(0))*1.0;
263
264   replace_stroke(0)(oldp-z1-z2);
265   set_botip(0,4,1);
266   set_botip(0,5,1);
267-268
269   tsu_accent.shift_anchors(ypart olda>vmetric(0.52))

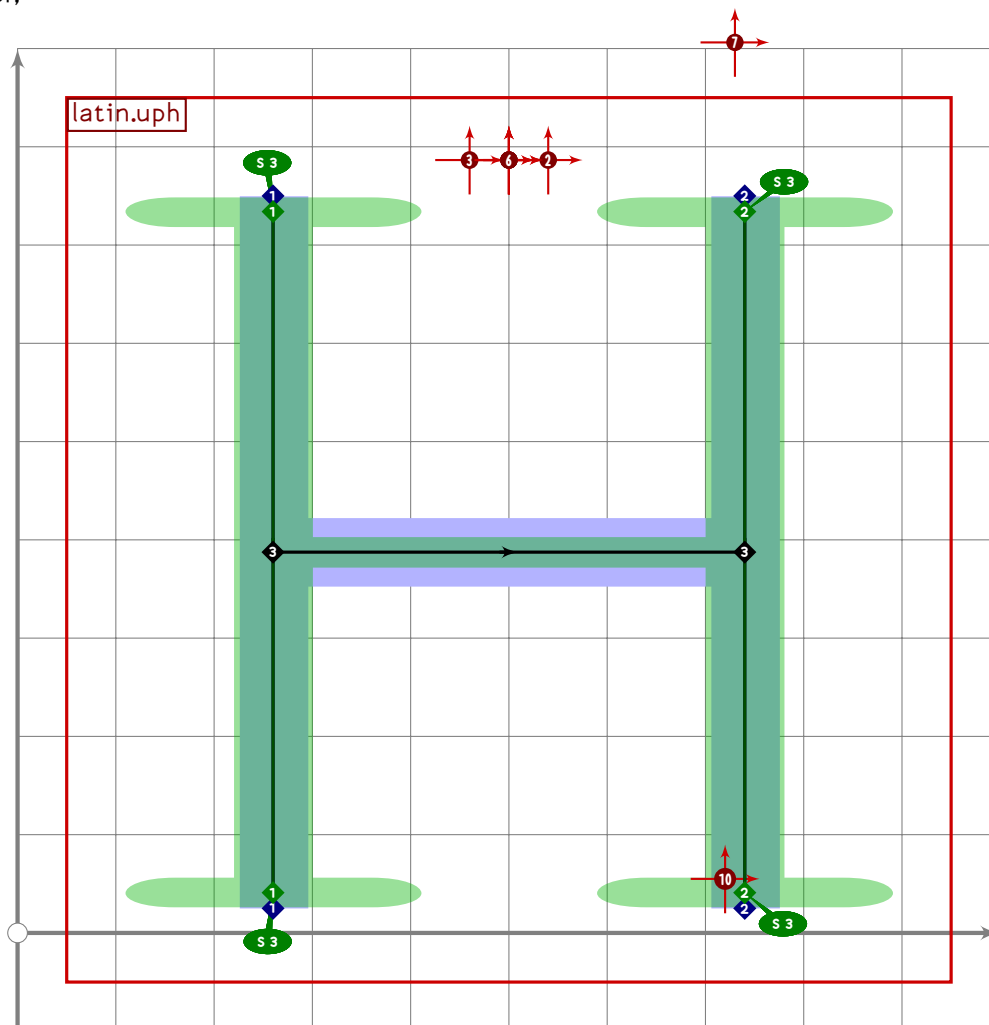
```

U+FF28  
tsuku.uniFF28

```

270    (((0,0) transformed tsu_xf.cap_upper_accent)-
271    ((0,0) transformed accent_default[anc_upper])+(44,0));
272    tsu_accent.shift_anchors(ypart olda<vmetric(0.52))((100,0));
273    expand_pbox;
274 enddef;

```



```

275
276 vardef latin.uph =
277   push_pbox_toexpand("latin.uph");
278   z1=(260,latin_wide_high_v);
279   z2=(740,latin_wide_high_v);
280   z3=(260,latin_wide_low_v);
281   z4=(740,latin_wide_low_v);
282
283   z5=whatever[z1,z3];
284   z6=whatever[z2,z4];
285   y5=y6=vmetric(0.5);
286
287   push_stroke(z1-z3,(1.6,1.6)-(1.6,1.6));
288   set_boserif(0,0,3);
289   set_boserif(0,1,3);

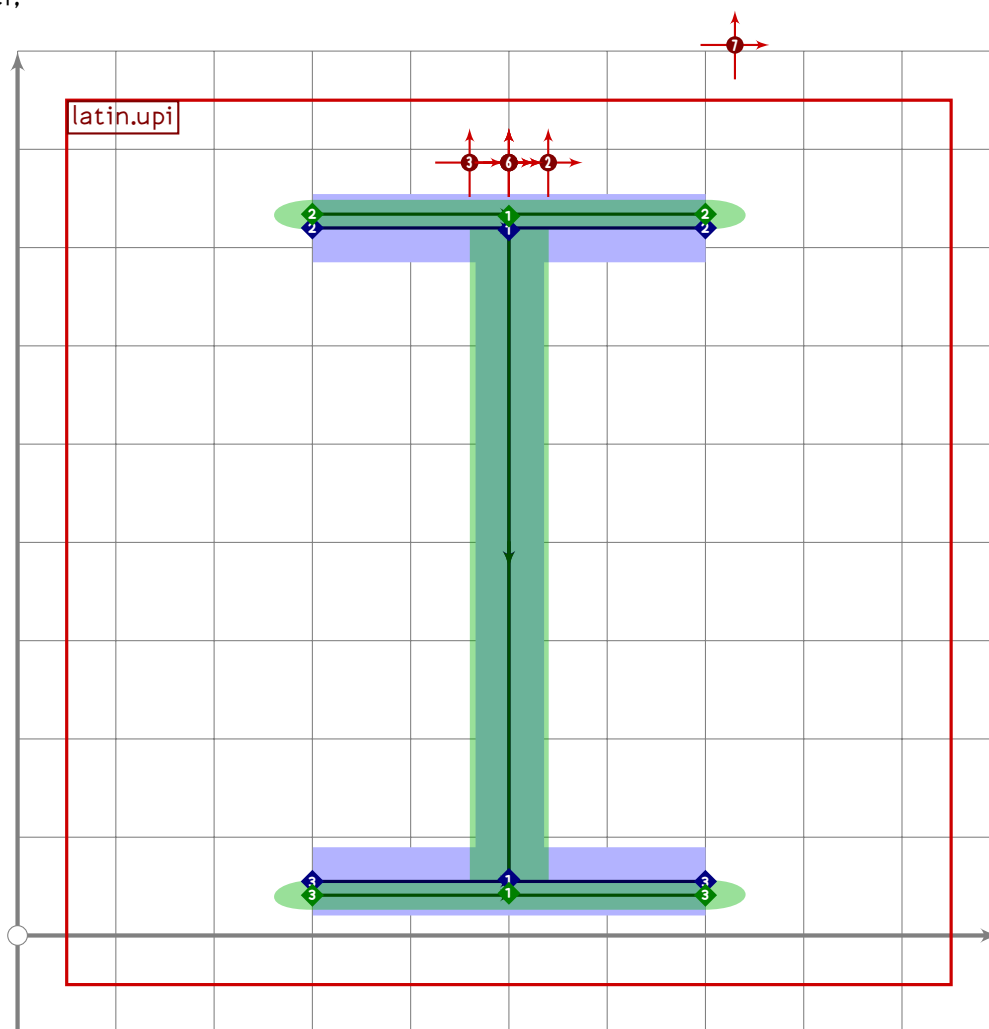
```

**LATI**

```

290
291 push_stroke(z2-z4,(1.6,1.6)-(1.6,1.6));
292 set_boserif(0,0,3);
293 set_boserif(0,1,3);
294
295 push_stroke(z5-z6,(1.6,1.6)-(1.6,1.6));
296
297 tsu_accent.shift_anchors(ypart olda>vmetric(0.52))
298   (((0,0) transformed tsu_xf.cap_upper_accent)-
299    ((0,0) transformed accent_default[anc_upper]));
300 tsu_accent.shift_anchors(ai=anc_lower_connect)((220,0));
301 expand_pbox;
302 enddef;

```



```

303
304 vardef latin.upi =
305   push_pbox_toexpand("latin.upi");
306   push_stroke((500,latin_wide_high_h-2)-(500,latin_wide_low_h+2),
307     (1.6,1.6)-(1.6,1.6));
308
309   push_stroke((300,latin_wide_high_h)-(700,latin_wide_high_h),

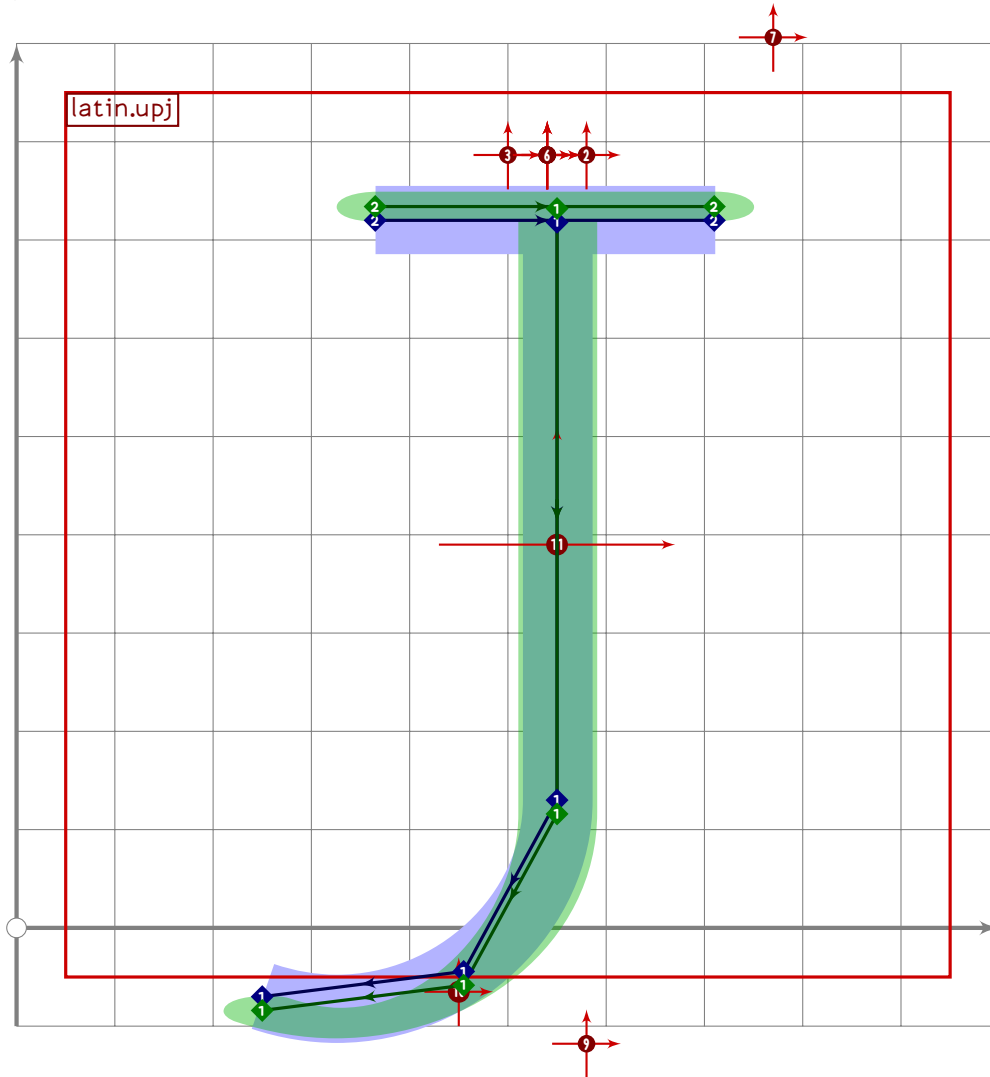
```

U+FF2A  
tsuku.uniFF2A

```

310 (1.6,1.6)–(1.6,1.6));
311
312 push_stroke((300,latin_wide_low_h)–(700,latin_wide_low_h),
313 (1.6,1.6)–(1.6,1.6));
314
315 tsu_accent.shift_anchors(ypart olda>vmetric(0.52))
316 (((0,0) transformed tsu_xf.cap_upper_accent)–
317 ((0,0) transformed accent_default[anc_upper]));
318 expand_pbox;
319 enddef;

```



**LATI**

```

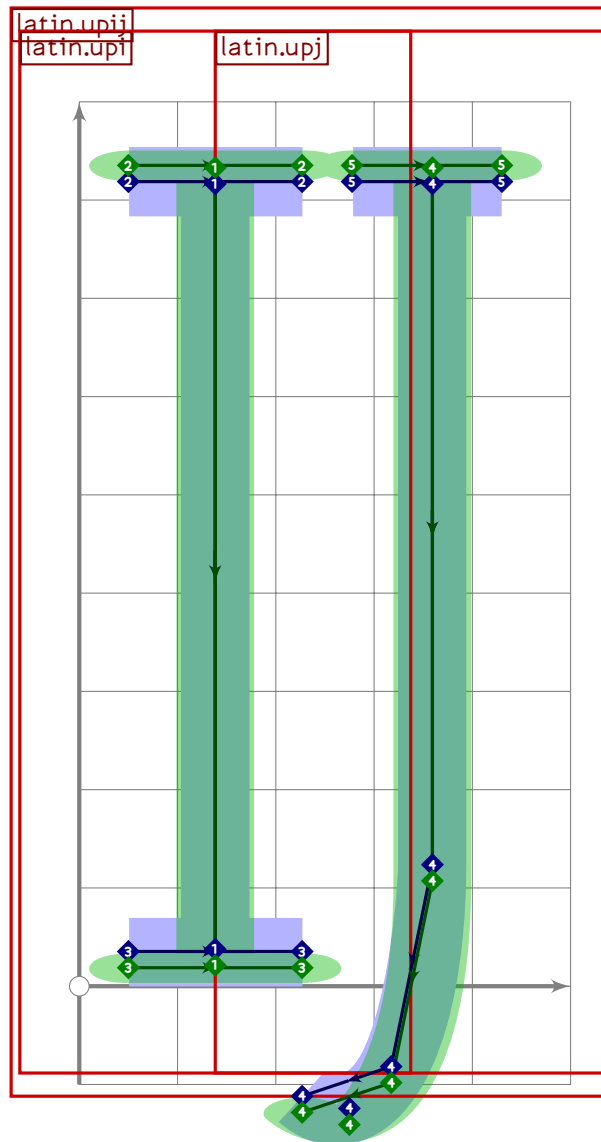
320
321 vardef latin.upj =
322   push_pbox_toexpand("latin.upj");
323   z1=(550,latin_wide_high_h-2);
324   z2=(550,latin_wide_low_h+75);
325   z3=z2+(-300,200);
326
327   push_stroke((z1–z2)..{curl 0.8}z3,

```

```

328 (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
329 replace_stroke(0)(insert_nodes(oldp)(1.5));
330
331 push_stroke(((365,latin_wide_high_h)-(710,latin_wide_high_h),
332 (1.6,1.6)-(1.6,1.6));
333
334 tsu_accent.shift_anchors(true)((50,0));
335
336 tsu_accent.shift_anchors(ypart olda>vmetric(0.52))
337 (((0,0) transformed tsu_xf.cap_upper_accent)-
338 ((0,0) transformed accent_default[anc_upper])+(-10,0));
339 tsu_accent.shift_anchors(ai=anc_lower)((30,0));
340 tsu_accent.shift_anchors(ai=anc_lower_connect)((-100,-120));
341 expand_pbox;
342 enddef;

```



```

343
344 vardef latin.upij =

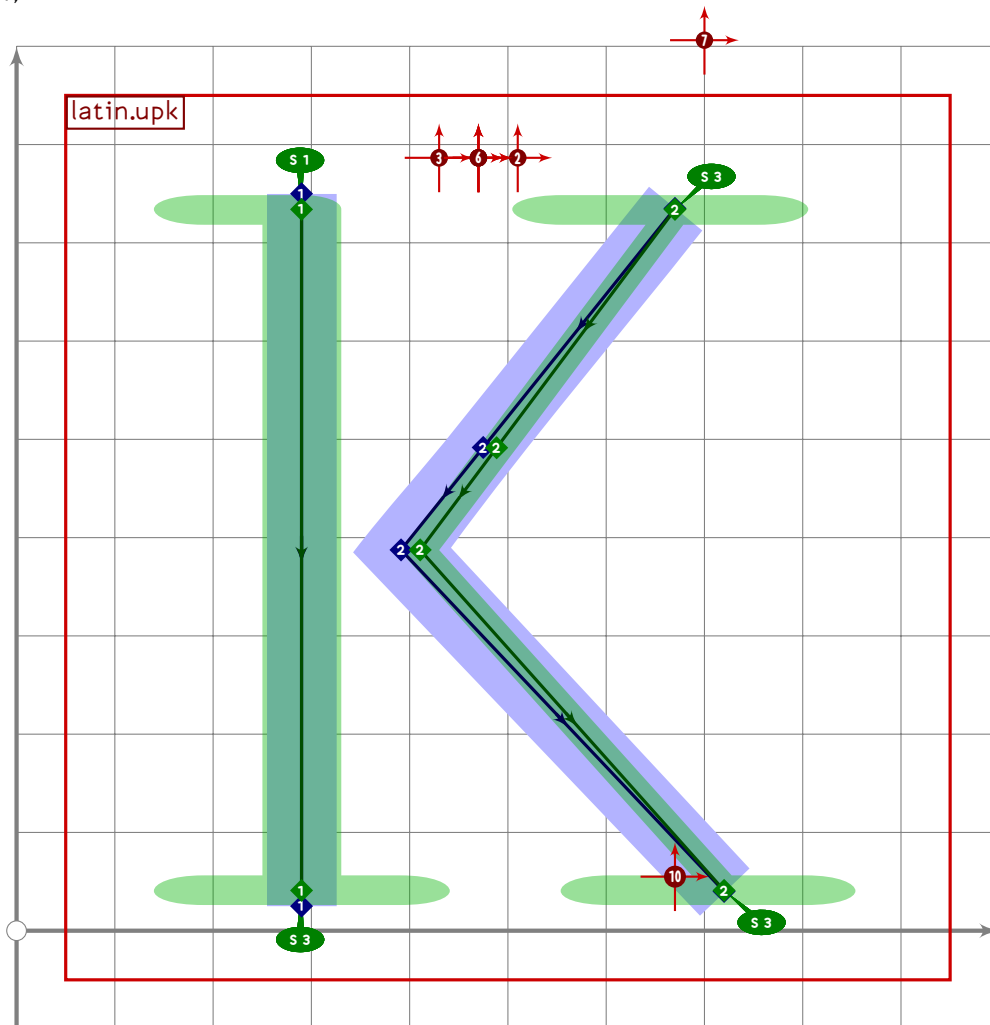
```

U+FF2B  
tsuku.uniFF2B

```

345 push_pbox_toexpand("latin.upij");
346 tsu_xform(identity shifted (-250,0))(latin.upi);
347 tsu_xform(identity shifted (200,0))(latin.upj);
348
349 tsu_accent.shift_anchors(ypart olda>vmetric(0.52))
350   (((0,0) transformed tsu_xf.cap_upper_accent)-
351    ((0,0) transformed accent_default[anc_upper]));
352 expand_pbox;
353 enddef;

```



```

354
355 vardef latin.upk =
356   push_pbox_toexpand("latin.upk");
357   z1=(290,latin_wide_high_v);
358   z2=(290,latin_wide_low_v);
359   z3=(670,0.5[latin_wide_high_h,latin_wide_high_v]);
360   x4=290+mbrush_width*if sharp_corners: 2.7 else: 2.3 fi;
361   y4=vmetric(0.5);
362   z5=(720,0.5[latin_wide_low_h,latin_wide_low_v]);
363
364   push_stroke(z1-z2,(1.6,1.6)-(1.6,1.6));

```

**LATI**

[illegible]

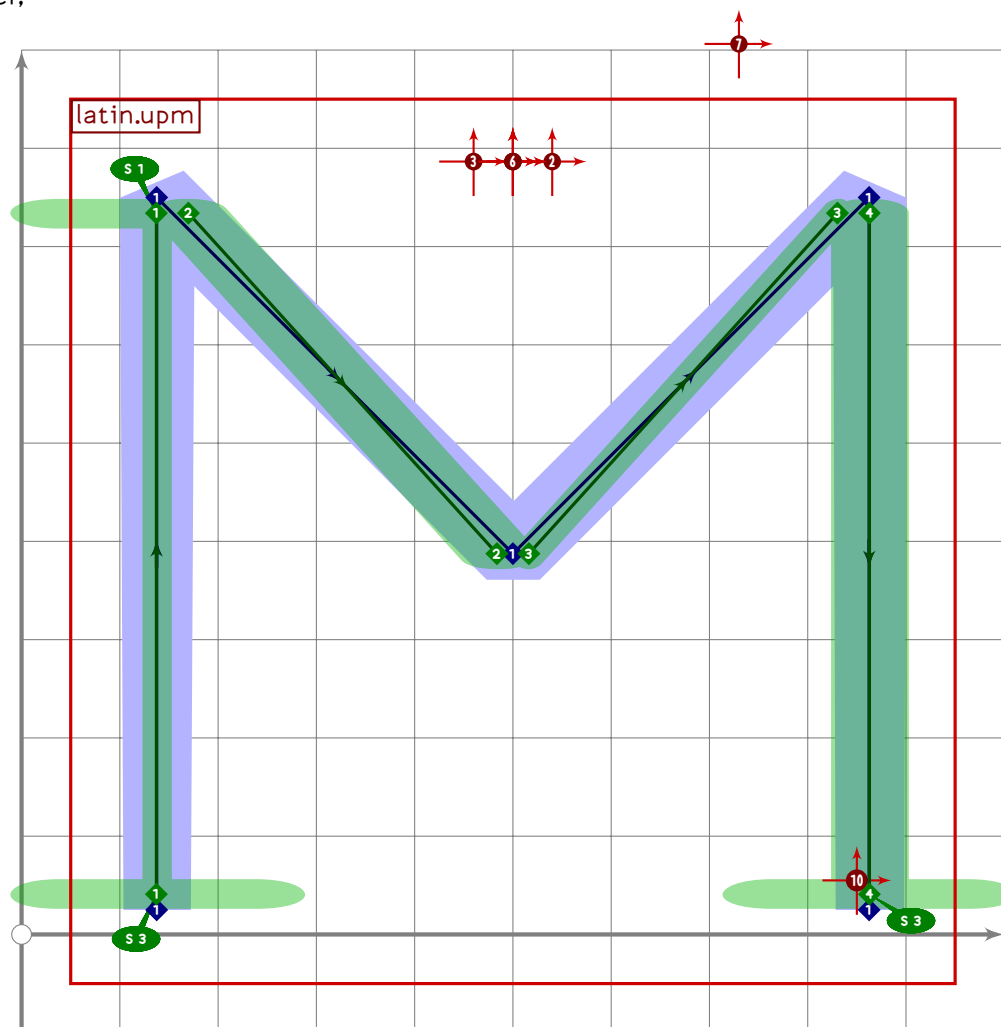
267

U+FF2D  
tsuku.uniFF2D

```

386 y1=latin_wide_high_v;
387 y2=y3=latin_wide_low_h;
388 x1=x2;
389 (x1+x3)/2=520;
390 (x3-x1)=(y1-y2)*0.6;
391
392 push_stroke(z1-z2-z3,(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
393 set_botip(0,1,1);
394 set_boserif(0,0,1);
395 set_boserif(0,1,3);
396
397 tsu_accent.shift_anchors((ypart olda>vmetric(0.52))
398                          and not (ai=anc_caron_comma))
399   (((0,0) transformed tsu_xf.cap_upper_accent)-
400    ((0,0) transformed accent_default[anc_upper]));
401 tsu_accent.shift_anchors(ai=anc_caron_comma)((-160,40));
402 expand_pbox;
403 enddef;

```



LATI

```

404
405 vardef latin.upm =

```

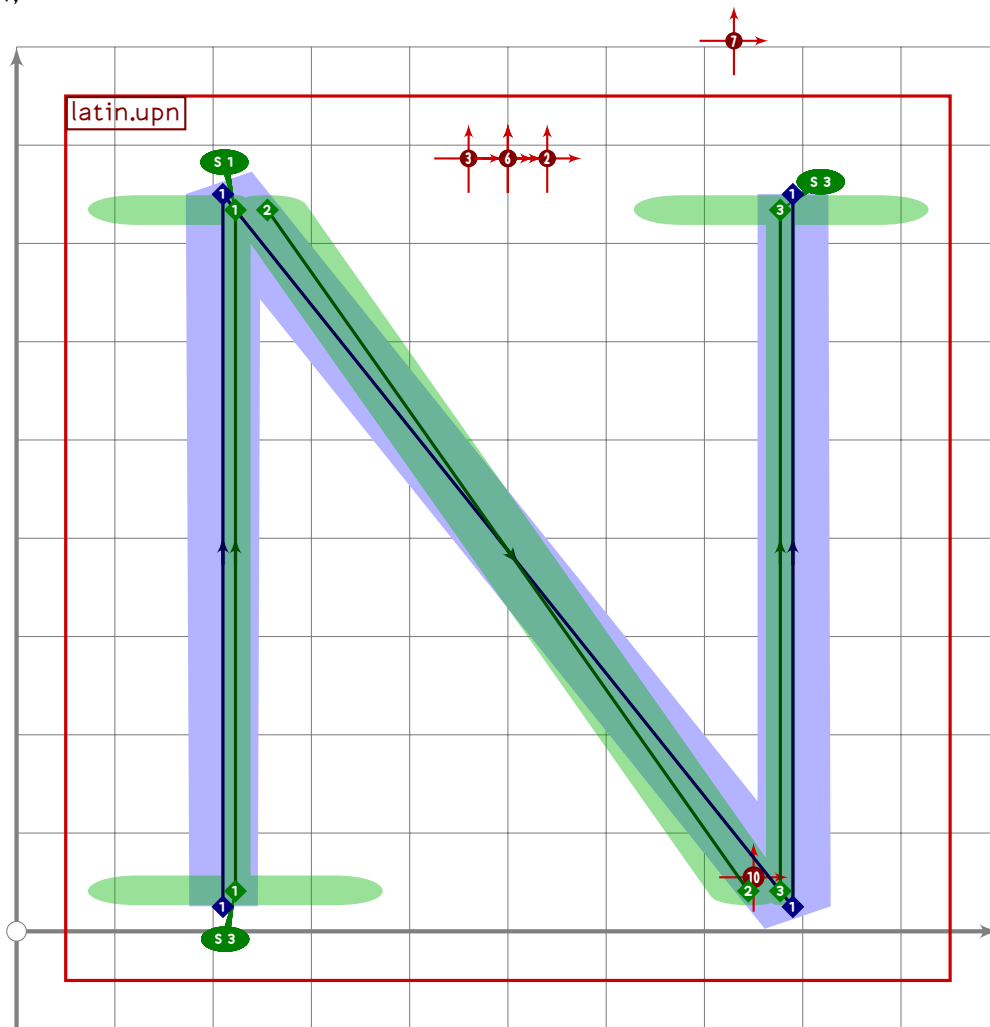
```

406 push_pbox_toexpand("latin.upm");
407 y1=y5=latin_wide_low_v;
408 y2=y4=latin_wide_high_v;
409 y3=(y1+y2)/2;
410
411 if do_alteration:
412     x1=x2;
413     x3=500+alternate_adjust/2;
414     x4=x5;
415     (x3-x1)=(x5-x3);
416
417     (x5-x1)=(y2-y1);
418
419     push_stroke((z1-z2) shifted (alternate_adjust*left),
420         (1.6,1.6)-(1.6,1.6));
421     set_boserif(0,0,3);
422     set_boserif(0,1,1);
423     set_boalternate(0);
424
425     push_stroke(z2-(z3+alternate_adjust*left),(1.6,1.6)-(1.6,1.6));
426
427     push_stroke(z3-(z4+alternate_adjust*left),(1.6,1.6)-(1.6,1.6));
428     set_boalternate(0);
429
430     push_stroke(z4-z5,(1.6,1.6)-(1.6,1.6));
431     set_boserif(0,1,3);
432 else:
433     x1=x2;
434     x3=500;
435     x4=x5;
436     (x3-x1)=(x5-x3);
437
438     (x5-x1)=(y2-y1);
439
440     push_stroke(z1-z2-z3-z4-z5,
441         (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
442     set_botip(0,1,0);
443     set_botip(0,2,0);
444     set_botip(0,3,0);
445     set_boserif(0,0,3);
446     set_boserif(0,1,1);
447     set_boserif(0,4,3);
448 fi;
449
450 tsu_accent.shift_anchors(y part olda>vmetric(0.52))
451     (((0,0) transformed tsu_xf.cap_upper_accent)-
452     ((0,0) transformed accent_default[anc_upper]));
453 tsu_accent.shift_anchors(ai=anc_lower_connect)((350,0));

```

U+FF2E  
tsuku.uniFF2E

```
454 expand_pbox;
455 enddef;
```



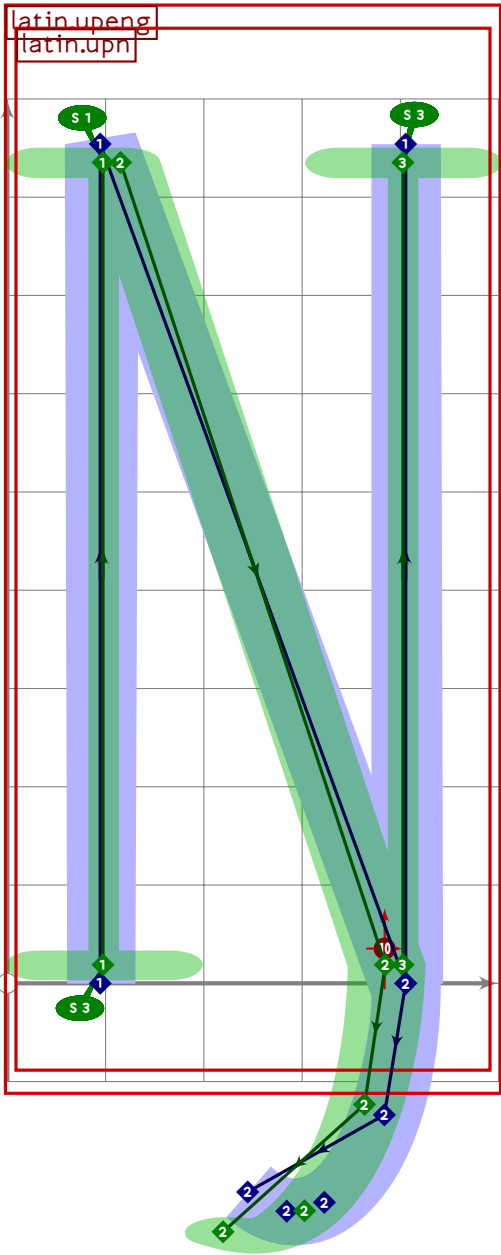
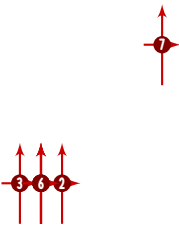
```
456
457 vardef latin.upn =
458   push_pbox_toexpand("latin.upn");
459   y1=y3=latin_wide_low_v;
460   y2=y4=latin_wide_high_v;
461
462   x1=x2;
463   x3=x4;
464   (x1+x3)/2=500;
465   (x3-x1)=(y2-y1)*4/5;
466
467   if do_alteration:
468     push_stroke(z1-z2,(1.6,1.6)-(1.6,1.6));
469     set_boserif(0,0,3);
470     set_boserif(0,1,1);
471     set_boalternate(0);
472
473   push_stroke((z2+alternate_adjust*right)-(z3+alternate_adjust*left),
```

**LATI**

```

474      (1.6,1.6)-(1.6,1.6));
475
476      push_stroke(z3-z4,(1.6,1.6)-(1.6,1.6));
477      set_boserif(0,1,3);
478      set_boalternate(0);
479  else:
480      push_stroke(z1-z2-z3-z4,(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
481      set_botip(0,1,0);
482      set_botip(0,2,0);
483      set_boserif(0,0,3);
484      set_boserif(0,1,1);
485      set_boserif(0,3,3);
486  fi;
487
488  tsu_accent.shift_anchors(ypart olda>vmetric(0.52))
489      (((0,0) transformed tsu_xf.cap_upper_accent)-
490      ((0,0) transformed accent_default[anc_upper]));
491  tsu_accent.shift_anchors(ai=anc_lower_connect)((250,0));
492  expand_pbox;
493 enddef;

```



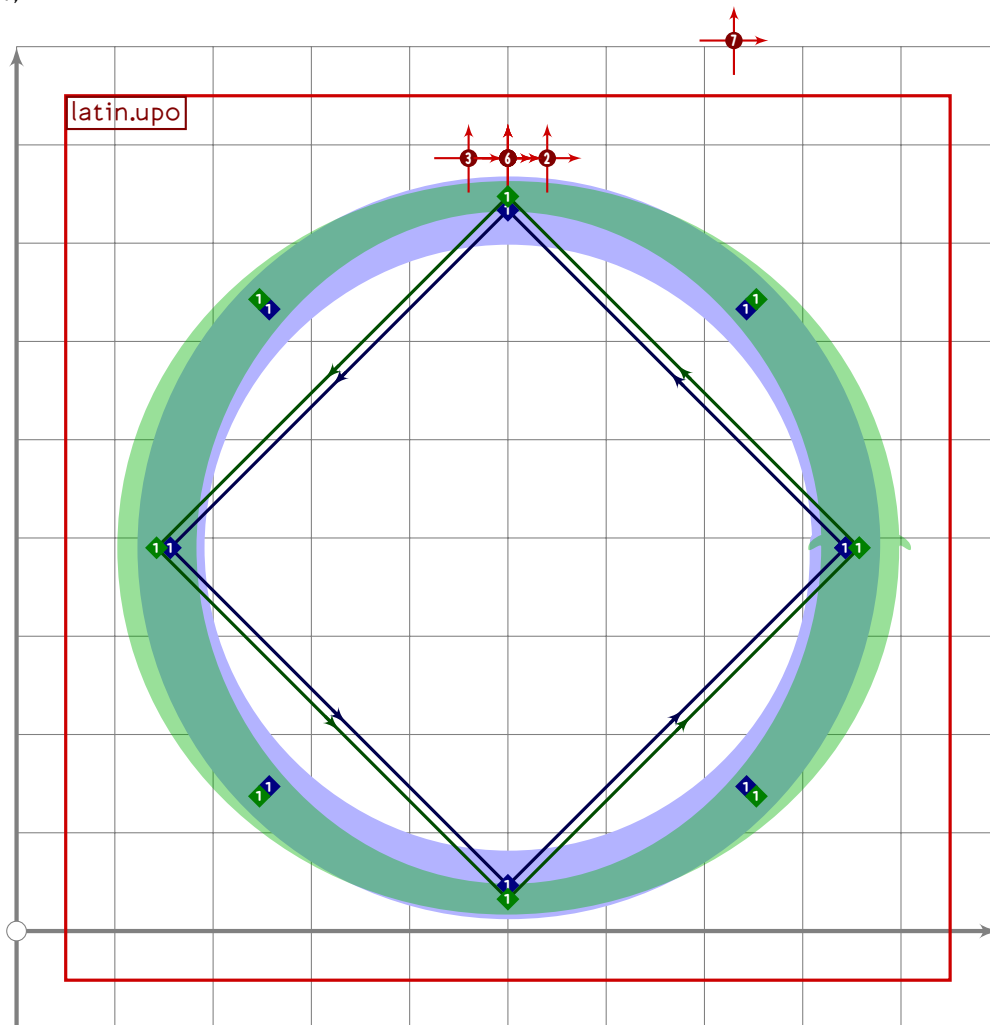
LATI

```
494  
495 vardef latin.upeng =  
496   push_pbox_toexpand("latin.upeng");  
497   latin.upn;  
498   y5=latin_wide_desc_h;
```

```

499 if do_alteration:
500     x5=x3-alternate_adjust-300;
501     replace_stroke(-1)(oldp{dir 268}{curl 0.8}z5);
502     replace_stroke(-1)(insert_nodes(oldp)(1.3));
503     replace_stroke(-1)(oldq-(1.6,1.6)-(1.6,1.6));
504 else:
505     x5=x3-300;
506     push_stroke(z3{dir 268}{curl 0.8}z5,(1.6,1.6)-(1.6,1.6));
507     replace_stroke(0)(insert_nodes(oldp)(0.3));
508 fi;
509
510 tsu_accent.shift_anchors(ypart olda>vmetric(0.52))
511   (((0,0) transformed tsu_xf.cap_upper_accent)-
512    ((0,0) transformed accent_default[anc_upper]));
513 expand_pbox;
514 enddef;

```



```

515
516 vardef latin.upo =
517   push_pbox_toexpand("latin.upo");
518   push_stroke(

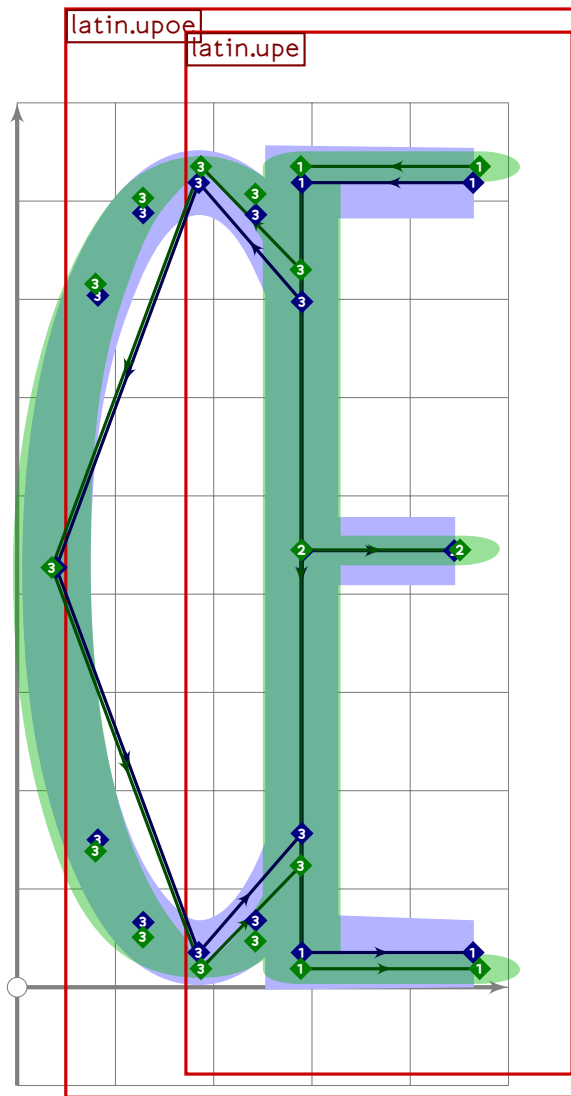
```

U+0152  
tsuku.OE

```

519 ((1,0)..(0,1)..(-1,0)..(0,-1)..cycle)
520 scaled ((latin_wide_high_r-latin_wide_low_r)/2)
521 shifted centre_pt,
522 (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-cycle);
523
524 tsu_accent.shift_anchors(ypart olda>vmetric(0.52))
525 (((0,0) transformed tsu_xf.cap_upper_accent)-
526 ((0,0) transformed accent_default[anc_upper]));
527 expand_pbox;
528 enddef;

```



**LATI**

```

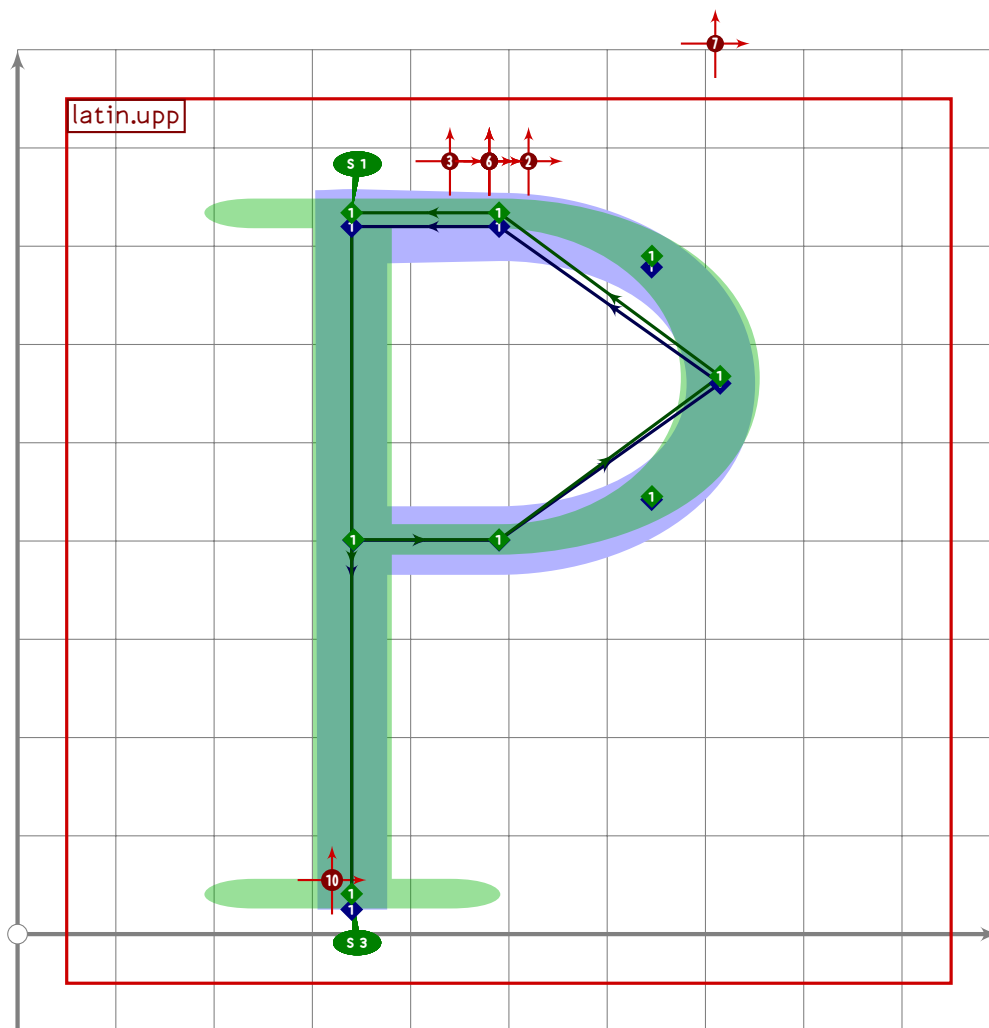
529
530 vardef latin.upoe =
531   push_pbox_toexpand("latin.upoe");
532   tsu_xform(identity shifted (280,0))(latin.upe);
533   set_boserif(-1,1,whatever);
534   set_boserif(-1,2,whatever);
535   push_stroke(
536     ((1,0)..(0,1)..(-1,0)..(0,-1)..(1,0))

```

```

537     scaled ((latin_wide_high_h-latin_wide_low_h)/2)
538     shifted (360,0.5[latin_wide_high_h,latin_wide_low_h]),
539     (1.2,1.2)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.2,1.2));
540 replace_strokep(0)(
541     subpath (xpart (oldp intersectiontimes get_strokep(-2)),
542             4-xpart ((reverse oldp) intersectiontimes get_strokep(-2)))
543     of oldp);
544
545 tsu_accent.shift_anchors(ypart olda>vmetric(0.52))
546     (((0,0) transformed tsu_xf.cap_upper_accent)-
547     ((0,0) transformed accent_default[anc_upper]));
548 expand_pbox;
549 enddef;
550
551 vardef latin.upp_base(expr b) =
552     x1=x5+2;
553     x5=340;
554     x2=x4=x5+b*0.4;
555     x3=x5+b;
556
557     y1=y2=vmetric(0.52);
558     y3=(y2+y4)/2;
559     y4=y5=latin_wide_high_h;
560
561     push_stroke(z1-z2{right}..z3..{left}z4-z5,
562     (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
563 enddef;

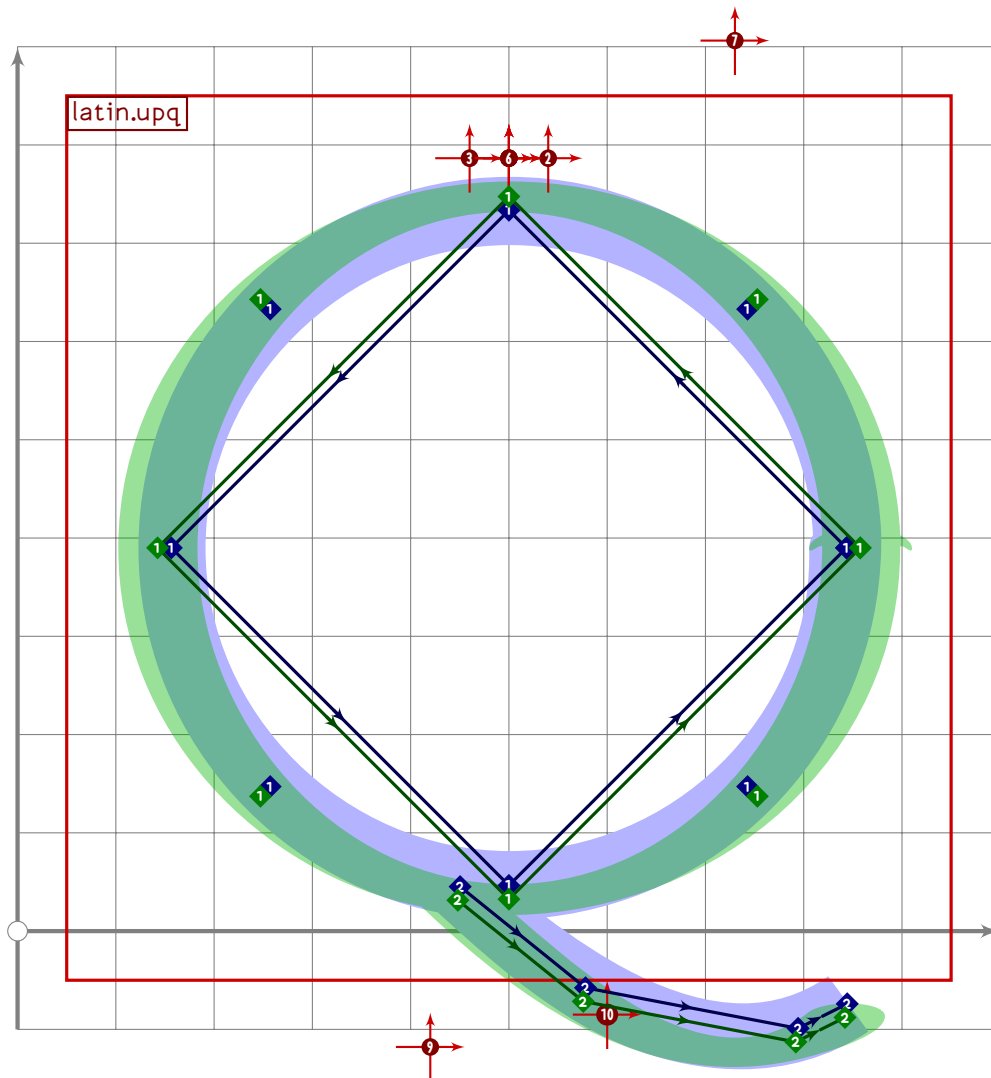
```



```

564
565 vardef latin.upp =
566   push_pbox_toexpand("latin.upp");
567   latin.upp_base(375);
568   replace_strokep(0)(oldp-(xpart point infinity of oldp,latin_wide_low_v));
569   replace_strokeq(0)(oldq-(1.6,1.6));
570   set_botip(0,4,1);
571   set_boserif(0,4,1);
572   set_boserif(0,5,3);
573
574   tsu_accent.shift_anchors(ypart olda>vmetric(0.52))
575     (((0,0) transformed tsu_xf.cap_upper_accent)-
576       ((0,0) transformed accent_default[anc_upper])+(-20,0));
577   tsu_accent.shift_anchors(ai=anc_lower_connect)((-180,0));
578   expand_pbox;
579 enddef;

```



```

580
581 vardef latin.upq =
582   push_pbox_toexpand("latin.upq");
583   push_stroke(((1,0)..(0,1)..(-1,0)..(0,-1)..cycle)
584     scaled ((latin_wide_high_r-latin_wide_low_r)/2)
585     shifted centre_pt,
586     (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-cycle);
587
588   z1=point 2.9 of get_strokep(0);
589   z2=z1+(350,-150);
590   z3=z2+(50,25);
591   push_stroke(subpath (0.02,2) of (z1{curl 0}..z2..z3),
592     (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
593   replace_strokep(0)(insert_nodes(oldp)(0.4));
594
595   tsu_accent.shift_anchors(ypart olda>vmetric(0.52))
596     (((0,0) transformed tsu_xf.cap_upper_accent)-
597     ((0,0) transformed accent_default[anc_upper]));
598   tsu_accent.shift_anchors(ai=anc_lower)((-80,0));

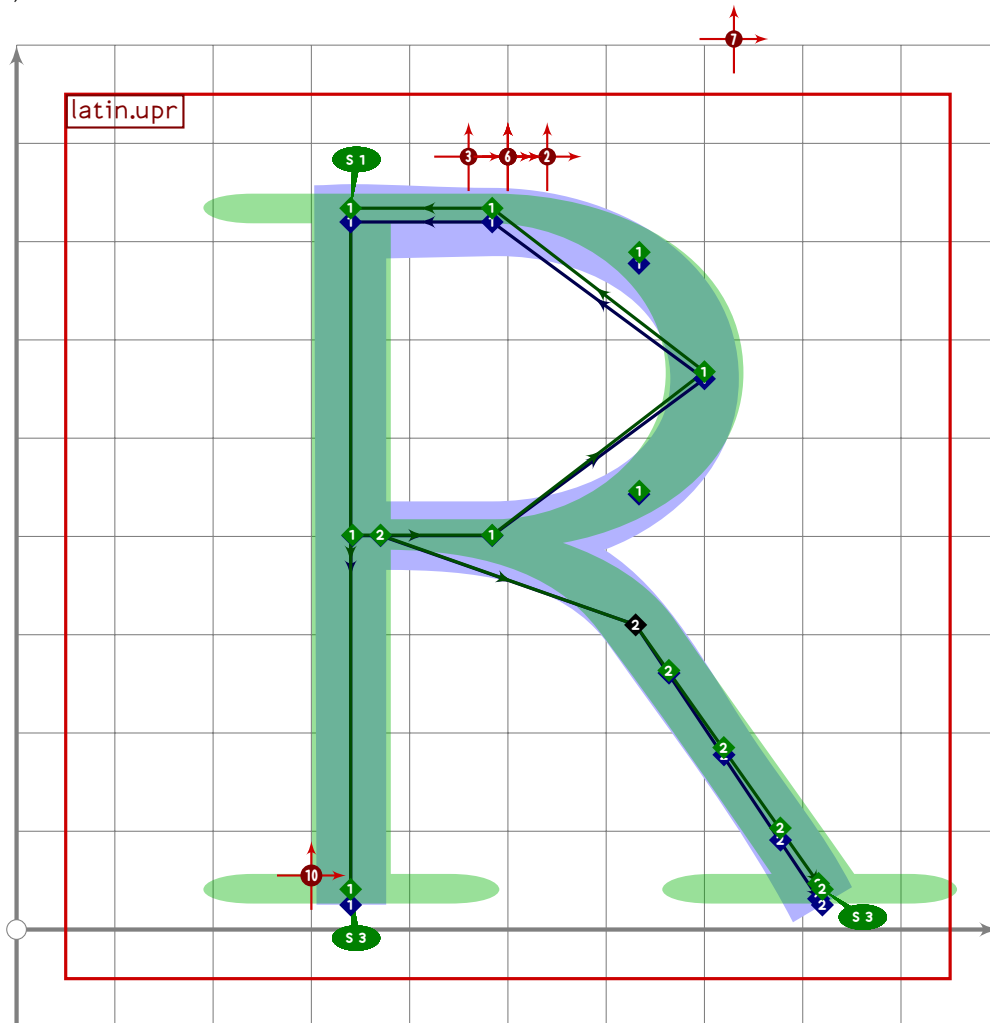
```

U+FF32  
tsuku.uniFF32

```

599 tsu_accent.shift_anchors(ai=anc_lower_connect)((100,-140));
600 expand_pbox;
601 enddef;

```



```

602
603 vardef latin.upr =
604   push_pbox_toexpand("latin.upr");
605   latin.upp_base(360);
606   replace_strokep(0)(oldp-(xpart point infinity of oldp,latin_wide_low_v));
607   replace_strokeq(0)(oldq-(1.6,1.6));
608   set_botip(0,4,1);
609   set_boserif(0,4,1);
610   set_boserif(0,5,3);
611
612   push_stroke((point 0.2 of get_strokep(0)){right}..(630,310)..
613     tension 3..(820,latin_wide_low_v),
614     (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
615   replace_strokep(0)(insert_nodes(oldp)(1.95));
616   set_boserif(0,3,3);
617
618   tsu_accent.shift_anchors(ypart olda>vmetric(0.52))

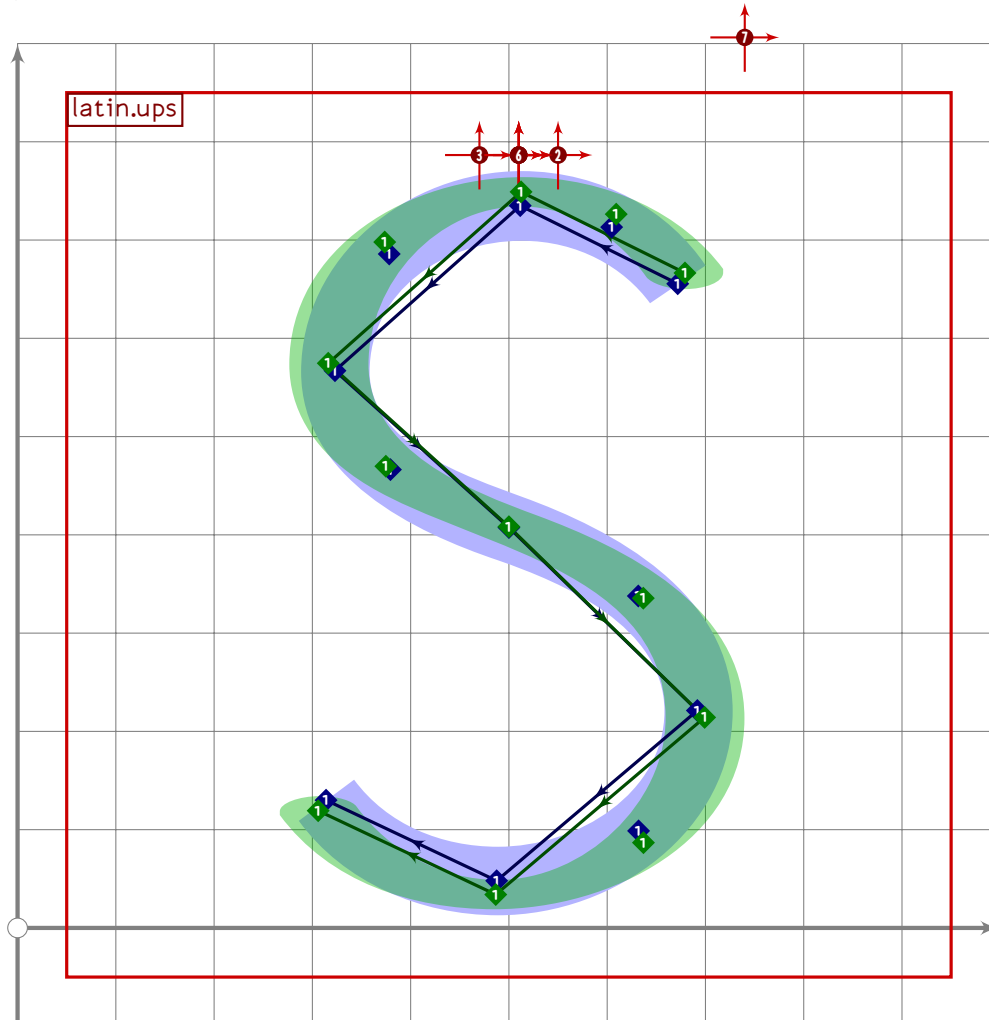
```

LATI

```

619    (((0,0) transformed tsu_xf.cap_upper_accent)-
620    ((0,0) transformed accent_default[anc_upper]));
621    tsu_accent.shift_anchors(ai=anc_lower_connect)((-200,0));
622    expand_pbox;
623 enddef;

```



```

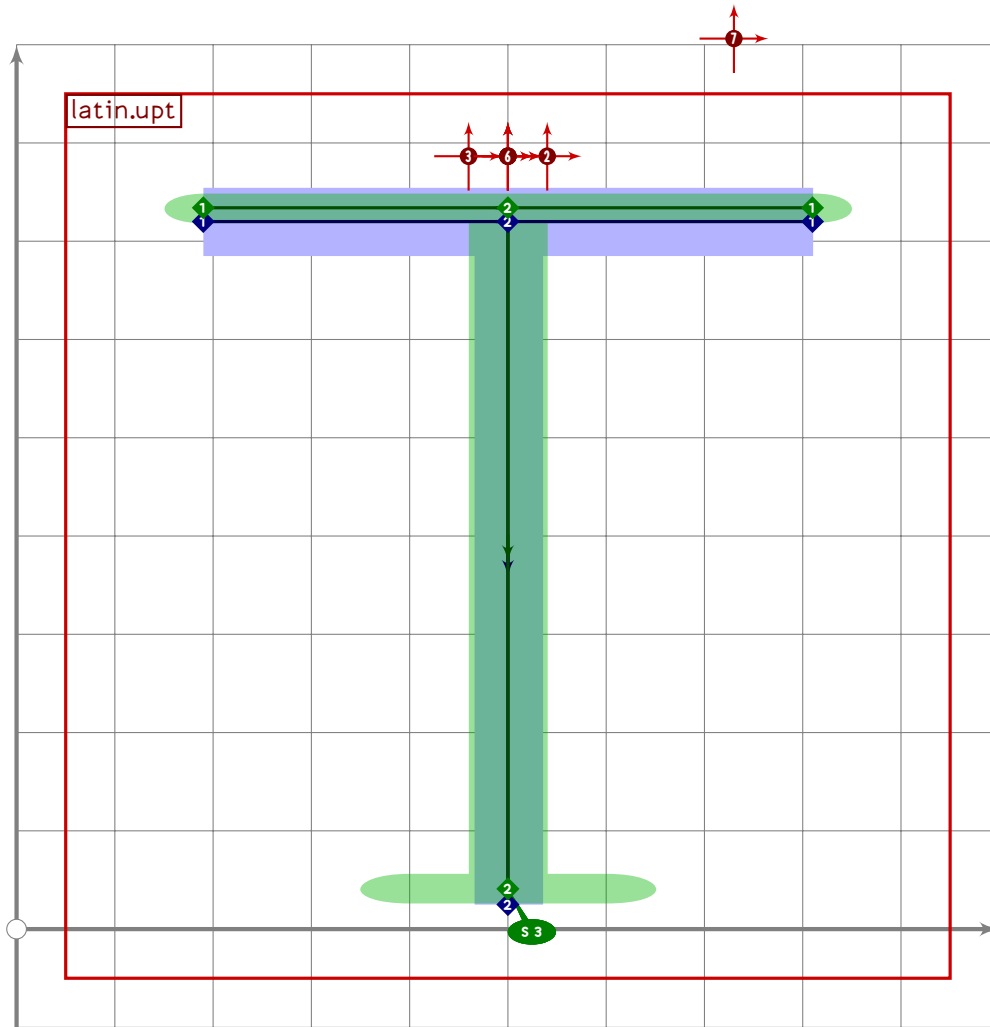
624
625 vardef latin.ups =
626   push_pbox_toexpand("latin.ups");
627   transform ta,tb;
628   path mycurve;
629
630   mycurve:=(1,0)..(0,1)..(-1,0);
631
632   y2=latin_wide_high_r;
633   y0=y3=vmetric(0.77);
634   y4=vmetric(0.53);
635   y5=y8=vmetric(0.25);
636   y6=latin_wide_low_r;
637
638   0.48[x1,x7]=0.48[x2,x6]=0.48[x3,x5]=x4=500;

```

```

639 x5-x1=20;
640 x5-x7=(y2-y6)*0.55;
641
642 (point 0 of mycurve) transformed ta=z0;
643 (point 0.35 of mycurve) transformed ta=z1;
644 (point 1 of mycurve) transformed ta=z2;
645 (point 2 of mycurve) transformed ta=z3;
646 xypart ta=0;
647
648 (point 0 of mycurve) transformed tb=z8;
649 (point 0.35 of mycurve) transformed tb=z7;
650 (point 1 of mycurve) transformed tb=z6;
651 (point 2 of mycurve) transformed tb=z5;
652
653 mycurve:=subpath (0.35,2) of mycurve;
654
655 push_stroke((mycurve transformed ta)..z4..(reverse mycurve transformed tb),
656   (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)
657   -(1.6,1.6));
658
659 tsu_accent.shift_anchors(ypart olda>vmetric(0.52))
660   (((0,0) transformed tsu_xf.cap_upper_accent)-
661   ((0,0) transformed accent_default[anc_upper])+(10,0));
662 expand_pbox;
663 enddef;

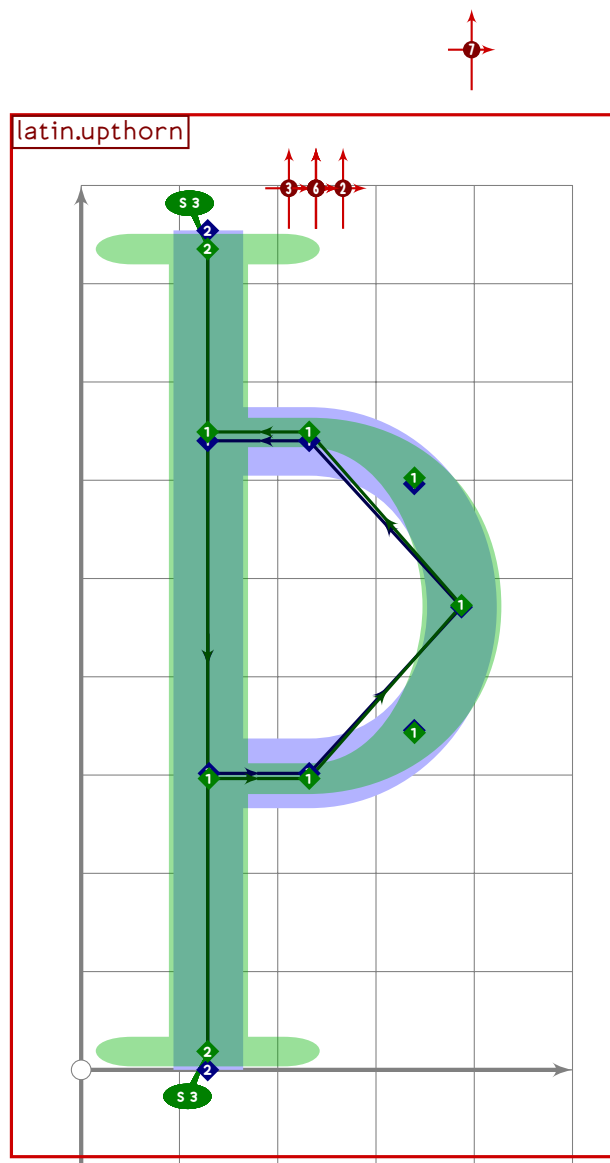
```



```

664
665 vardef latin.upt =
666   push_pbox_toexpand("latin.upt");
667   z1=(190,latin_wide_high_h);
668   z2=(500,latin_wide_high_h);
669   z3=(810,latin_wide_high_h);
670   z4=(500,latin_wide_low_v);
671
672   push_stroke(z1-z3,(1.6,1.6)-(1.6,1.6));
673
674   push_stroke(z2-z4,(1.6,1.6)-(1.6,1.6));
675   set_boserif(0,1,3);
676
677   tsu_accent.shift_anchors(ypart olda>vmetric(0.52))
678     (((0,0) transformed tsu_xf.cap_upper_accent)-
679     ((0,0) transformed accent_default[anc_upper]));
680   expand_pbox;
681 enddef;

```

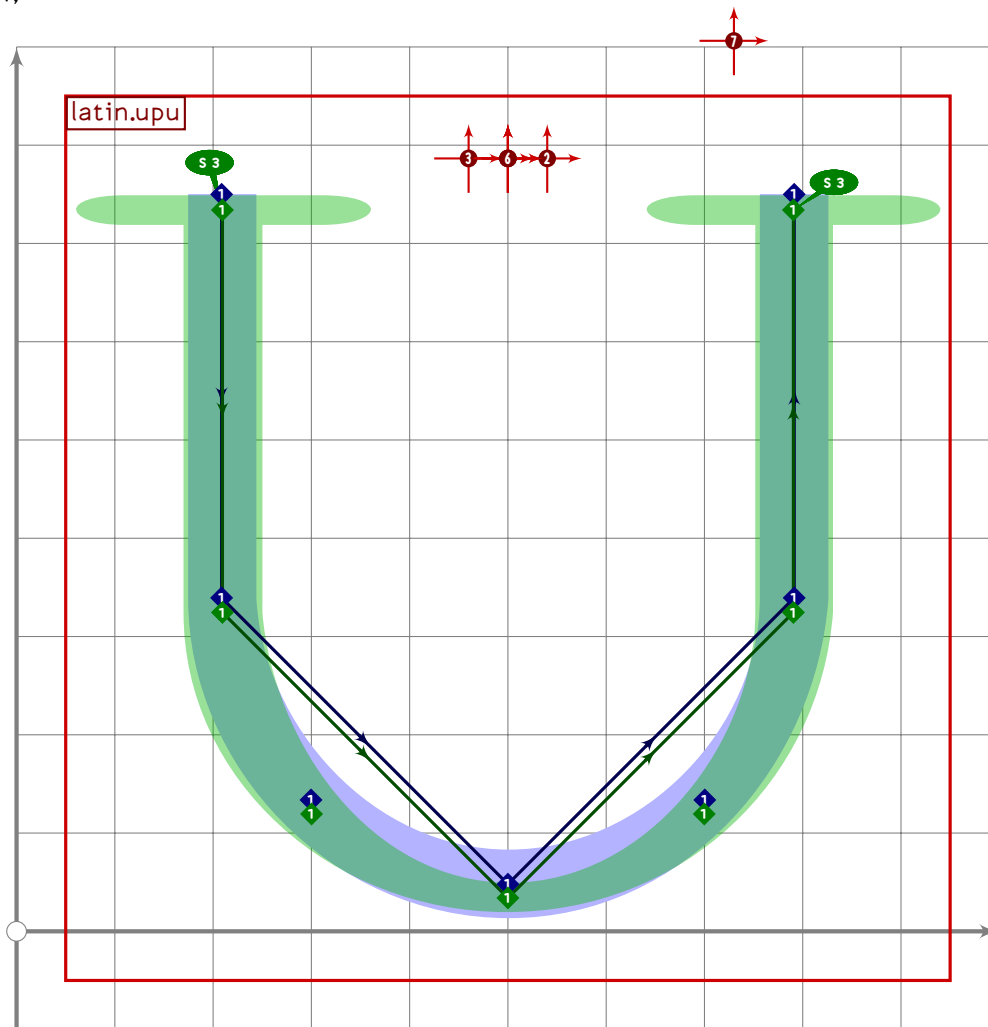


```

682
683 vardef latin.upthorn =
684   push_pbox_toexpand("latin.upthorn");
685   latin.upp_base(375);
686   replace_strokep(0)(oldp
687     shifted (-llcorner oldp) yscaled 0.9
688     shifted ((llcorner oldp)+(0,vmetric(0)-vmetric(0.18))));
689   push_stroke((xpart point infinity of get_strokep(0),latin_wide_high_v)
690     -(xpart point infinity of get_strokep(0),latin_wide_low_v),
691     (1.6,1.6)-(1.6,1.6));
692   set_boserif(0,0,3);
693   set_boserif(0,1,3);
694
695   tsu_accent.shift_anchors(ypart olda>vmetric(0.52))
696     (((0,0) transformed tsu_xf.cap_upper_accent)-
697     ((0,0) transformed accent_default[anc_upper]));

```

```
698     expand_pbox;
699 enddef;
```



```

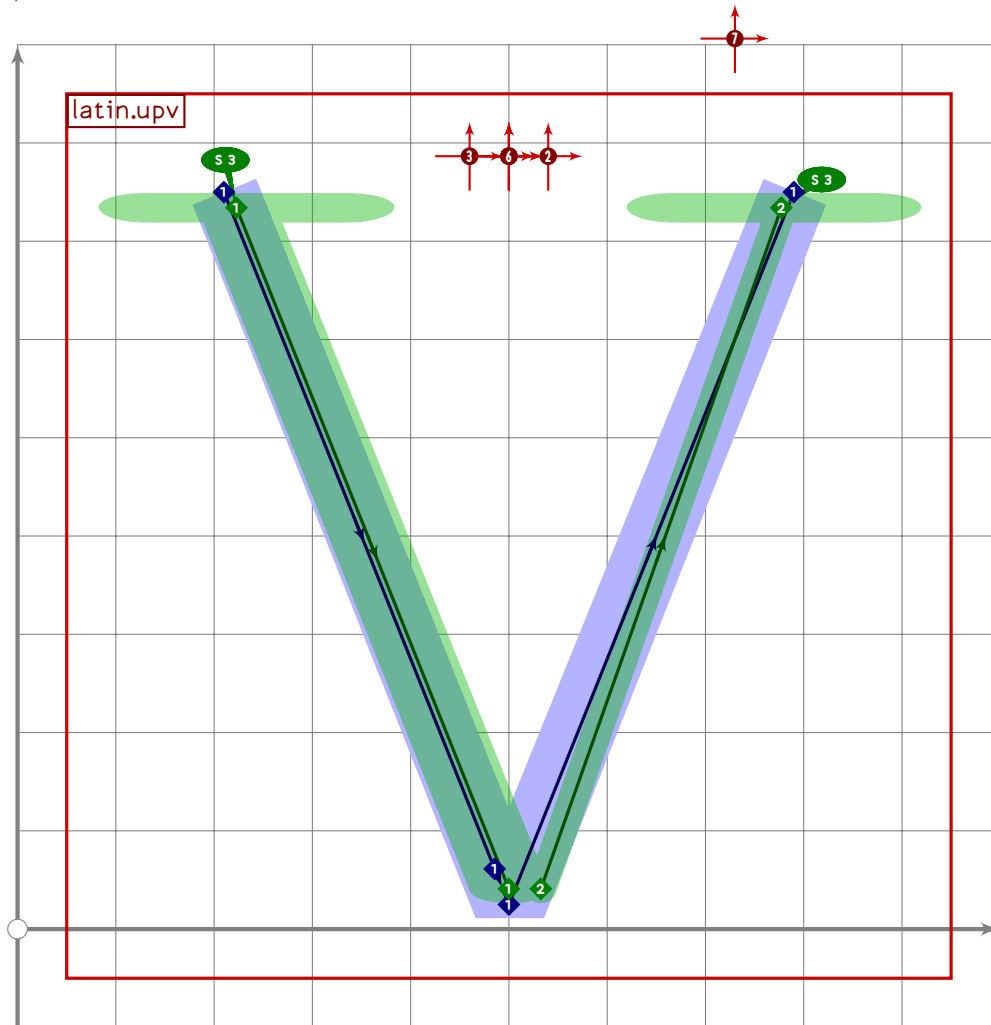
700
701 vardef latin.upu =
702     push_pbox_toexpand("latin.upu");
703     x1=x2;
704     x3=500;
705     x4=x5;
706     (x1+x5)/2=x3;
707     (x5-x1)=(y1-y3)*0.83;
708
709     y1=y5=latin_wide_high_v;
710     y2=y4;
711     y2-y3=x3-x2;
712     y3=latin_wide_low_r;
713
714     push_stroke(z1-z2{dir 274}..z3.{dir 86}z4-z5,
715         (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
716     set_boserif(0,0,3);
717     set_boserif(0,4,3);

```

```

718
719 tsu_accent.shift_anchors(ypart olda>vmetric(0.52))
720 (((0,0) transformed tsu_xf.cap_upper_accent)-
721 ((0,0) transformed accent_default[anc_upper]));
722 expand_pbox;
723 enddef;

```



```

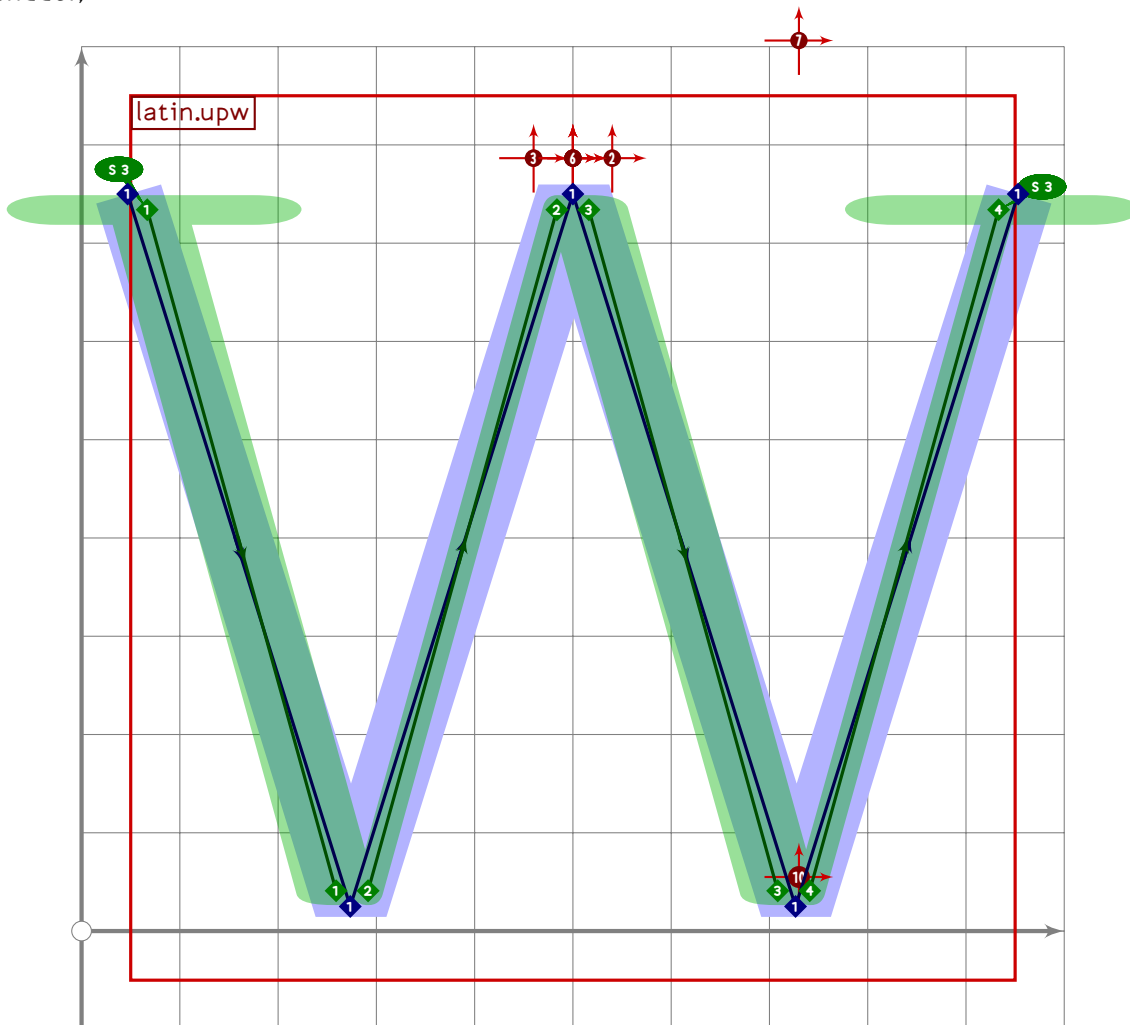
724
725 vardef latin.upv =
726   push_pbox_toexpand("latin.upv");
727   (x1+x3)/2=x2=500;
728
729   y1=y3=latin_wide_high_v;
730   y2=latin_wide_low_v;
731
732   (x3-x1)=(y1-y2)*0.8;
733
734   if do_alteration:
735     push_stroke(z1-z2,(1.6,1.6)-(1.6,1.6));
736     set_boserif(0,0,3);
737

```

```

738   push_stroke((z2+alternate_adjust*right)-z3,(1.6,1.6)-(1.6,1.6));
739   set_boserif(0,1,3);
740   set_boalternate(0);
741   else:
742     push_stroke(z1-(0.95[z1,z2])-z2-z3,
743       (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
744     set_botip(0,2,0);
745     set_boserif(0,0,3);
746     set_boserif(0,3,3);
747   fi;
748
749   tsu_accent.shift_anchors(ypart olda>vmetric(0.52))
750     (((0,0) transformed tsu_xf.cap_upper_accent)-
751       ((0,0) transformed accent_default[anc_upper]));
752   expand_pbox;
753   enddef;

```



```

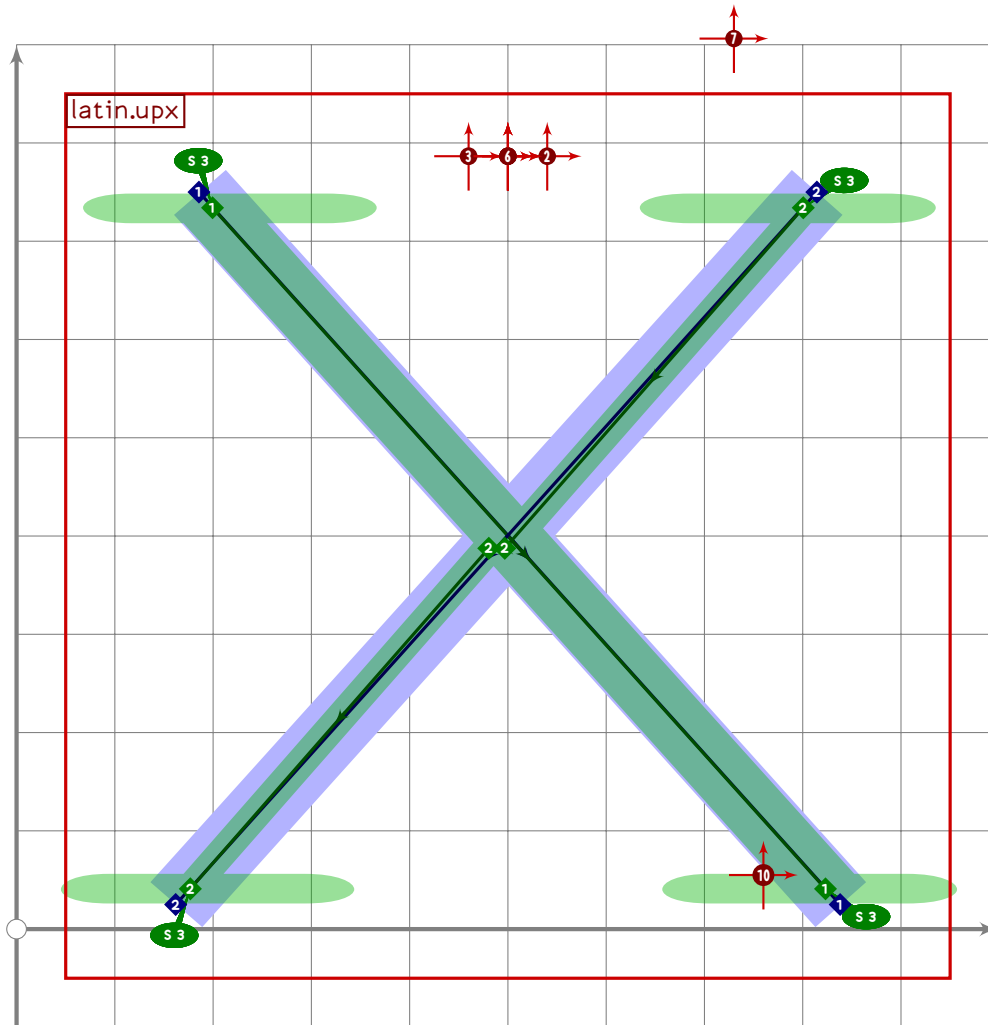
754
755   vardef latin.upw =
756     push_pbox_toexpand("latin.upw");
757     if do_alteration:

```

```

758 (x1+x5)/2=(x2+x4)/2=x3=500-alternate_adjust/2;
759 (x3-x2)=(x2-x1);
760
761 y1=y3=y5=latin_wide_high_v;
762 y2=y4=latin_wide_low_v;
763
764 (x5-x1)=(y1-y2)*1.25-(3*alternate_adjust);
765
766 push_stroke((z1-z2) shifted (alternate_adjust*left),
767 (1.6,1.6)-(1.6,1.6));
768 set_boserif(0,0,3);
769
770 push_stroke(z2-z3,(1.6,1.6)-(1.6,1.6));
771 set_boalternate(0);
772
773 push_stroke((z3-z4) shifted (alternate_adjust*right),
774 (1.6,1.6)-(1.6,1.6));
775
776 push_stroke((z4-z5) shifted (alternate_adjust*right*2),
777 (1.6,1.6)-(1.6,1.6));
778 set_boserif(0,1,3);
779 set_boalternate(0);
780 else:
781 (x1+x5)/2=(x2+x4)/2=x3=500;
782 (x3-x2)=(x2-x1);
783
784 y1=y3=y5=latin_wide_high_v;
785 y2=y4=latin_wide_low_v;
786
787 (x5-x1)=(y1-y2)*1.25;
788
789 push_stroke(z1-z2-z3-z4-z5,
790 (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
791 set_botip(0,1,0);
792 set_botip(0,2,0);
793 set_botip(0,3,0);
794 set_boserif(0,0,3);
795 set_boserif(0,4,3);
796 fi;
797
798 tsu_accent.shift_anchors(ypart olda>vmetric(0.52))
799 (((0,0) transformed tsu_xf.cap_upper_accent)-
800 ((0,0) transformed accent_default[anc_upper]));
801 tsu_accent.shift_anchors(ai=anc_lower_connect)((230,0));
802 expand_pbox;
803 enddef;

```



```

804
805 vardef latin.upx =
806   push_pbox_toexpand("latin.upx");
807   (x1+x3)/2=500;
808   (x2+x4)/2=500;
809   (x2+x3-x1-x4)=((y1-y2)*0.9)*2;
810   (x3-x1)=(x2-x4)*0.93;
811
812   y1=y3=latin_wide_high_v;
813   y2=y4=latin_wide_low_v;
814
815   push_stroke(z1-z2,(1.6,1.6)-(1.6,1.6));
816   set_boserif(0,0,3);
817   set_boserif(0,1,3);
818
819   if do_alteration:
820     push_stroke(z3-(0.5[z3,z4]+alternate_adjust*right/4)
821       -(0.5[z3,z4]+alternate_adjust*left/4)-z4,
822       (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
823     set_boserif(0,0,3);

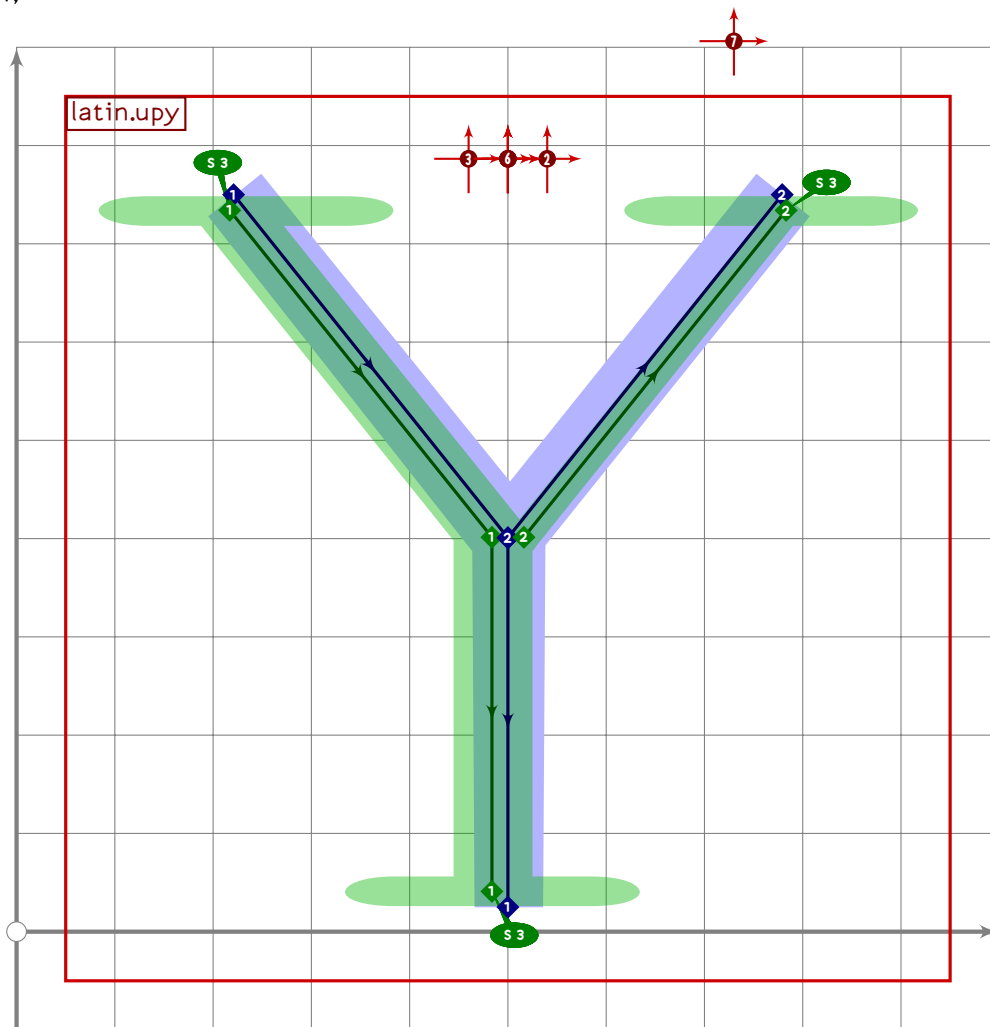
```

U+FF39  
tsuku.uniFF39

```

824   set_boserif(0,3,3);
825   else:
826     push_stroke(z3-z4,(1.6,1.6)-(1.6,1.6));
827     set_boserif(0,0,3);
828     set_boserif(0,1,3);
829   fi;
830   set_boalternate(0);
831
832   tsu_accent.shift_anchors(ypart olda>vmetric(0.52))
833     (((0,0) transformed tsu_xf.cap_upper_accent)-
834     ((0,0) transformed accent_default[anc_upper]));
835   tsu_accent.shift_anchors(ai=anc_lower_connect)((260,0));
836   expand_pbox;
837 enddef;

```



LATI

```

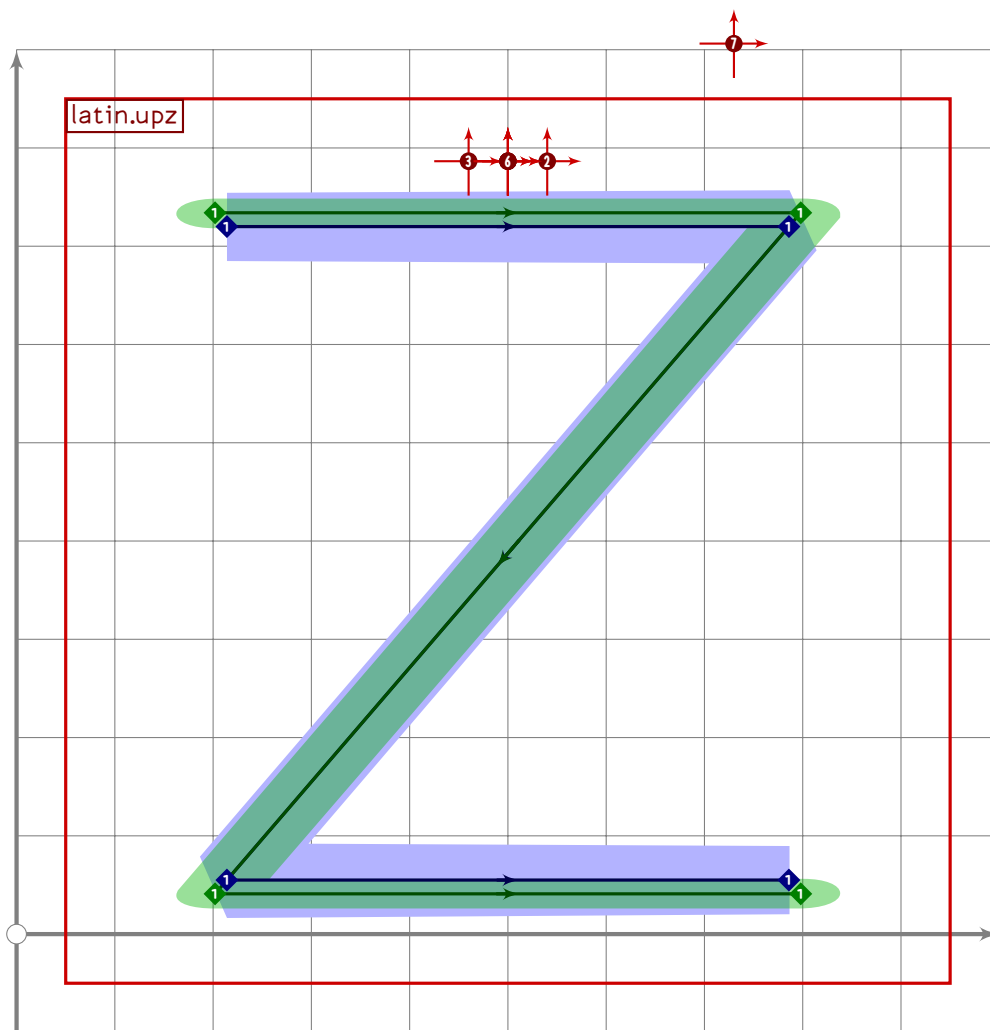
838
839 vardef latin.upy =
840   push_pbox_toexpand("latin.upy");
841   (x3+x1)/2=x2=x4=if do_alteration: 500-alternate_adjust/2 else: 500 fi;
842   (x3-x1)=0.77*(y1-y4);
843

```

```

844 y1=y3=latin_wide_high_v;
845 y2=vmetric(0.52);
846 y4=latin_wide_low_v;
847
848 push_stroke(z1-z2-z4,(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
849 set_botip(0,1,0);
850 set_boserif(0,0,3);
851 set_boserif(0,2,3);
852
853 if do_alteration:
854     push_stroke((z2-z3) shifted (alternate_adjust*right),
855         (1.6,1.6)-(1.6,1.6));
856 else:
857     push_stroke(z2-z3,(1.6,1.6)-(1.6,1.6));
858 fi;
859 set_boserif(0,1,3);
860 set_boalterate(0);
861
862 tsu_accent.shift_anchors(y part olda>vmetric(0.52))
863     (((0,0) transformed tsu_xf.cap_upper_accent)-
864     ((0,0) transformed accent_default[anc_upper]));
865 expand_pbox;
866 enddef;

```



```

867
868 vardef latin.upz =
869   push_pbox_toexpand("latin.upz");
870   y1=y2=latin_wide_high_h;
871   y3=y4=latin_wide_low_h;
872
873   x1=x3;
874   x2=x4;
875   (x1+x2)/2=500;
876   (x2-x1)=(y1-y3)*0.86;
877
878   push_stroke(z1-z2-z3-z4,(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
879   set_botip(0,1,0);
880   set_botip(0,2,0);
881
882   tsu_accent.shift_anchors(ypart olda>vmetric(0.52))
883     (((0,0) transformed tsu_xf.cap_upper_accent)-
884      ((0,0) transformed accent_default[anc_upper]));
885   expand_pbox;
886 enddef;

```

```

887
888
889
890 vardef latin.double_lowa =
891   push_pbox_toexpand("latin.double_lowa");
892   x1=(-0.12)[x6,x3];
893   x3=x4=x5=x8;
894   0.51[x6,x3]=500;
895   x3-x1=0.82*(y2-y4);
896   x2=0.63[x6,x3];
897   x7=0.36[x6,x3];
898
899   y1=0.7[y4,y2];
900   y3=0.77[y4,y2];
901   y2=latin_wide_xheight_r;
902   y4=latin_wide_low_v;
903   y5=0.68[y4,y2];
904   y6=0.32[y7,y5];
905   y7=latin_wide_low_h;
906   y8=0.3[y4,y2];
907
908   push_stroke(z1{curl 0.2}..z2{right}..z3{down}..z4,
909     (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
910   replace_strokep(0)(subpath (0.2,3) of oldp);
911   set_boserif(0,3,2);
912
913   push_stroke(z5{dir 240}..z6{down}..z7{right}..z8,
914     (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
915   replace_strokep(0)(subpath (0,2.97) of oldp);
916   replace_strokep(0)(insert_nodes(oldp)(0.5));
917
918   tsu_accent.shift_anchors(true)((12,0));
919   tsu_accent.shift_anchors(ai=anc_ring)((28,0));
920   expand_pbox;
921 enddef;
922
923 vardef latin.single_lowa =
924   push_pbox_toexpand("latin.single_lowa");
925   (x1+x4)/2=520;
926   (x1-x4)=(y3-y1)*0.81;
927   x1=x6-8;
928   x3=0.4[x1,x4];
929   x5=0.36[x4,x1];
930
931   y1=latin_wide_low_v;
932   y3=latin_wide_xheight_h;
933   y4=0.47[y5,y3];
934   y5=latin_wide_low_h;

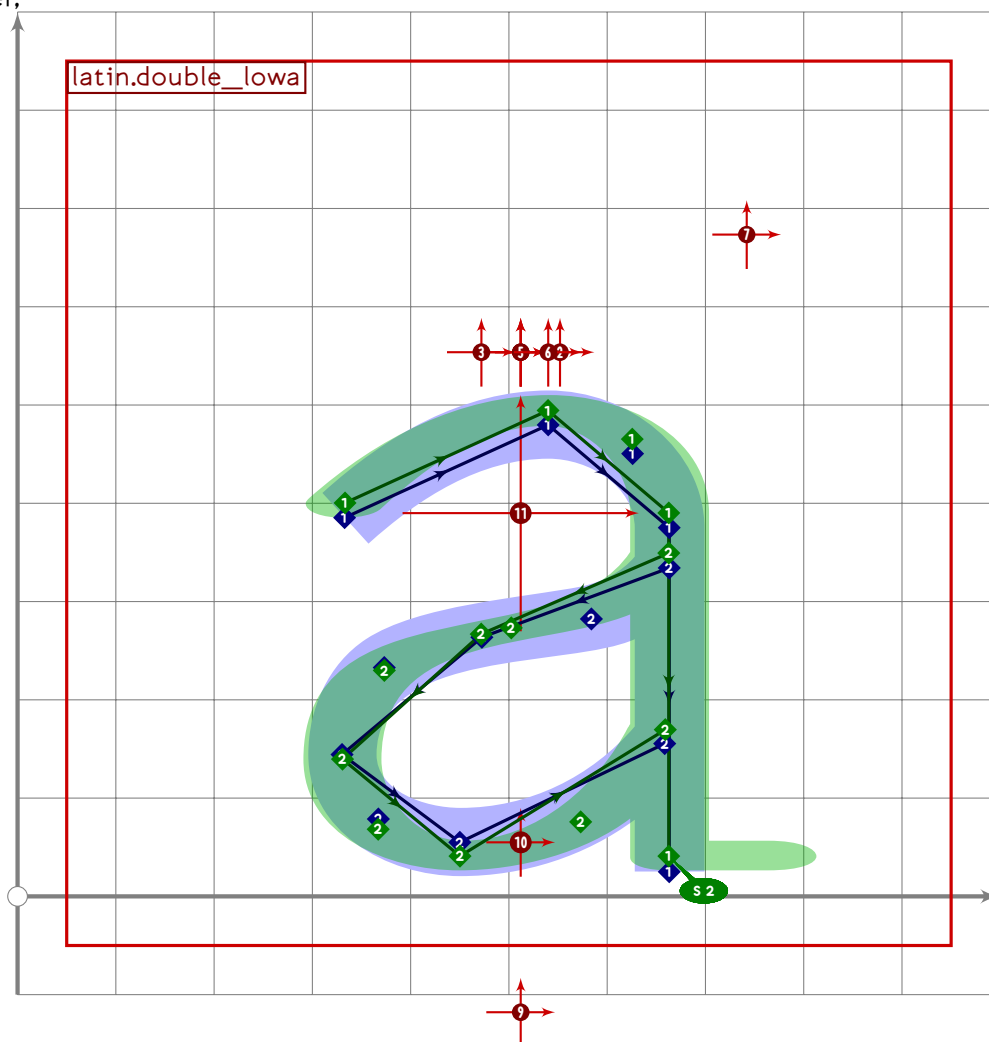
```

U+FF41  
tsuku.uniFF41

```

935 y6=0.37[y1,y3];
936
937 z7=(x1,y3);
938 z2=0.3[z7,0.5[z3,z1]];
939
940 push_stroke(interpath(0.5)
941   (z1{up}{z7}{left}{left}{z3}{down}{z4}{right}{z5..z6,
942     z1{up}{z2}{left}{z3}{down}{z4}{right}{z5..z6},
943   (1.3,1.3)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-
944     (1.6,1.6)-(1.6,1.6)-(1.6,1.6));
945 replace_strokep(0)(reverse(insert_nodes(olddp)(0.5)));
946 set_boserif(0,6;if do_italic_hook: 11 else: 2 fi);
947 set_botip(0,4,1);
948 expand_pbox;
949 enddef;

```

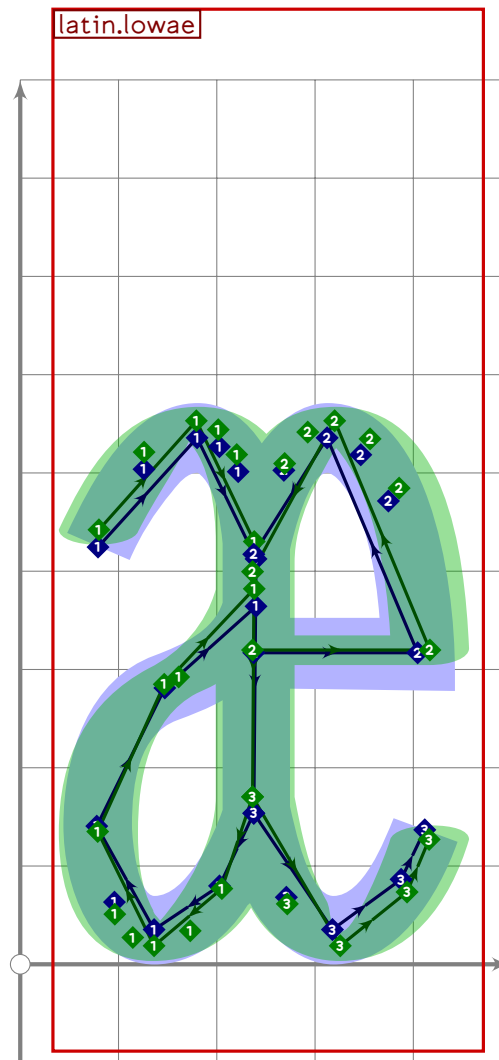


LATI

```

950
951 vardef latin.low_a =
952   latin.double_lowa;
953 enddef;

```



```

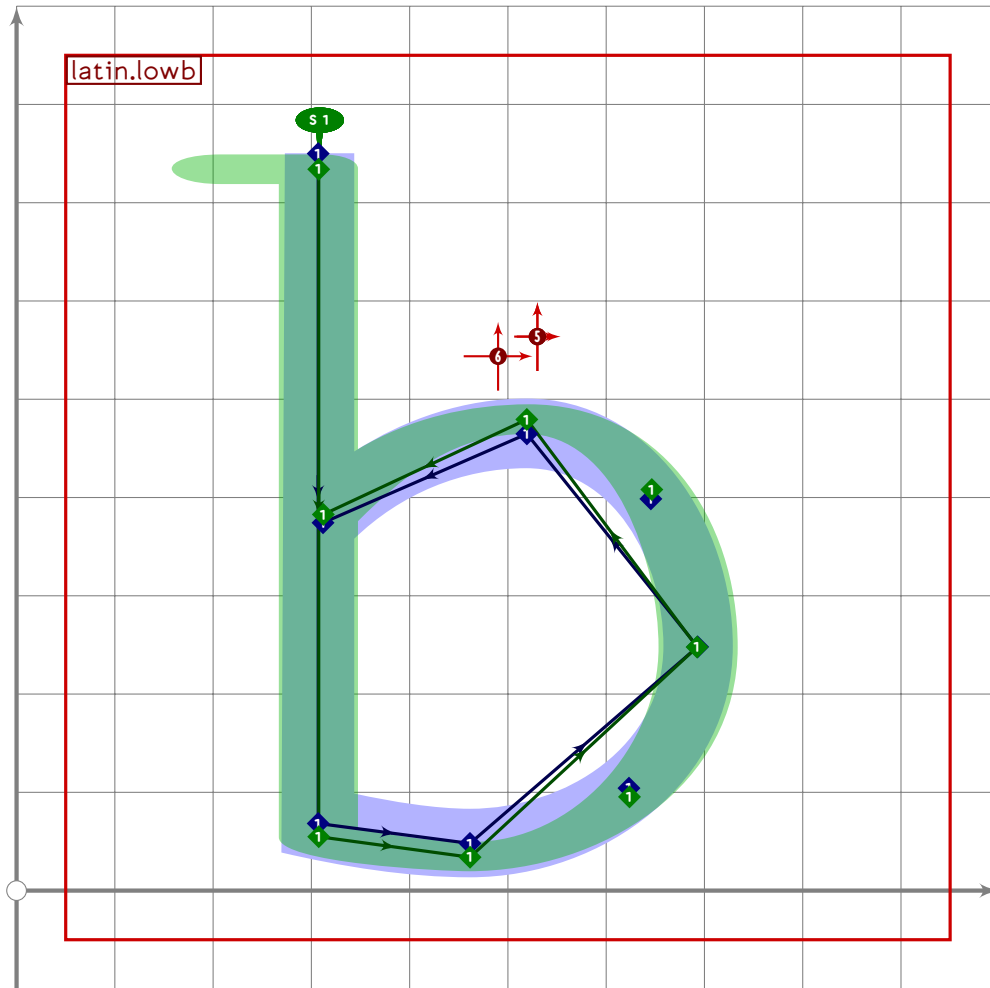
954
955 vardef latin.lowae =
956   push_pbox_toexpand("latin.lowae");
957   begingroup
958     save saved_sp;
959     save astem,abowl,astroke,estroke,etop,ebot;
960     path astem,abowl,astroke,estroke,etop,ebot;
961     saved_sp:=sp;
962
963     latin.double_lowae;
964     set_boserif(-1,3,whatever);
965     astem:=get_stroke(-1);
966     abowl:=get_stroke(0);
967
968     numeric x[],y[];
969     latin.lowe;
970     estroke:=get_stroke(0);
971
972     for i=saved_sp upto sp-1:

```

```

973     obstacktype[i]:=otnull;
974 endfor;
975
976 astroke:=(subpath (0,2) of astem)-reverse abowl;
977
978 numeric x[],y[];
979 z1=point 3 of astroke;
980 path xbp;
981 xbp=estroke shifted (0,ypart ((llcorner astroke)-(llcorner estroke)));
982 x2=xpart (xbp intersectionpoint ((470,y1)-(0,y1)));
983 astroke:=astroke shifted (470-x1,0);
984
985 estroke:=estroke shifted (470-x2,0);
986 xbp:=xbp shifted (470-x2,0);
987
988 ebot:=subpath (xpart (xbp intersectiontimes ((500,y1)-(0,y1))),
989               infinity) of xbp;
990 ebot:=insert_nodes(ebot)((length ebot)-0.3);
991
992 etop:=
993     subpath (ypart (((470,1000)-(470,0))
994               intersectiontimes (subpath (0,1) of estroke)),
995             1+ypart (((470,1000)-(470,0))
996               intersectiontimes (subpath (1,infinity) of estroke)))
997     of estroke;
998
999 astroke:=insert_nodes(astroke)(3,4);
1000
1001 push_stroke(astroke,
1002     (1.6,1.6) for i=1 upto length astroke: -(1.6,1.6) endfor);
1003 push_stroke(etop,
1004     (1.6,1.6) for i=1 upto length etop: -(1.6,1.6) endfor);
1005 set_botip(0,1);
1006 push_stroke(ebot,
1007     (1.6,1.6) for i=1 upto length ebot: -(1.6,1.6) endfor);
1008 endgroup;
1009 expand_pbox;
1010 enddef;

```



```

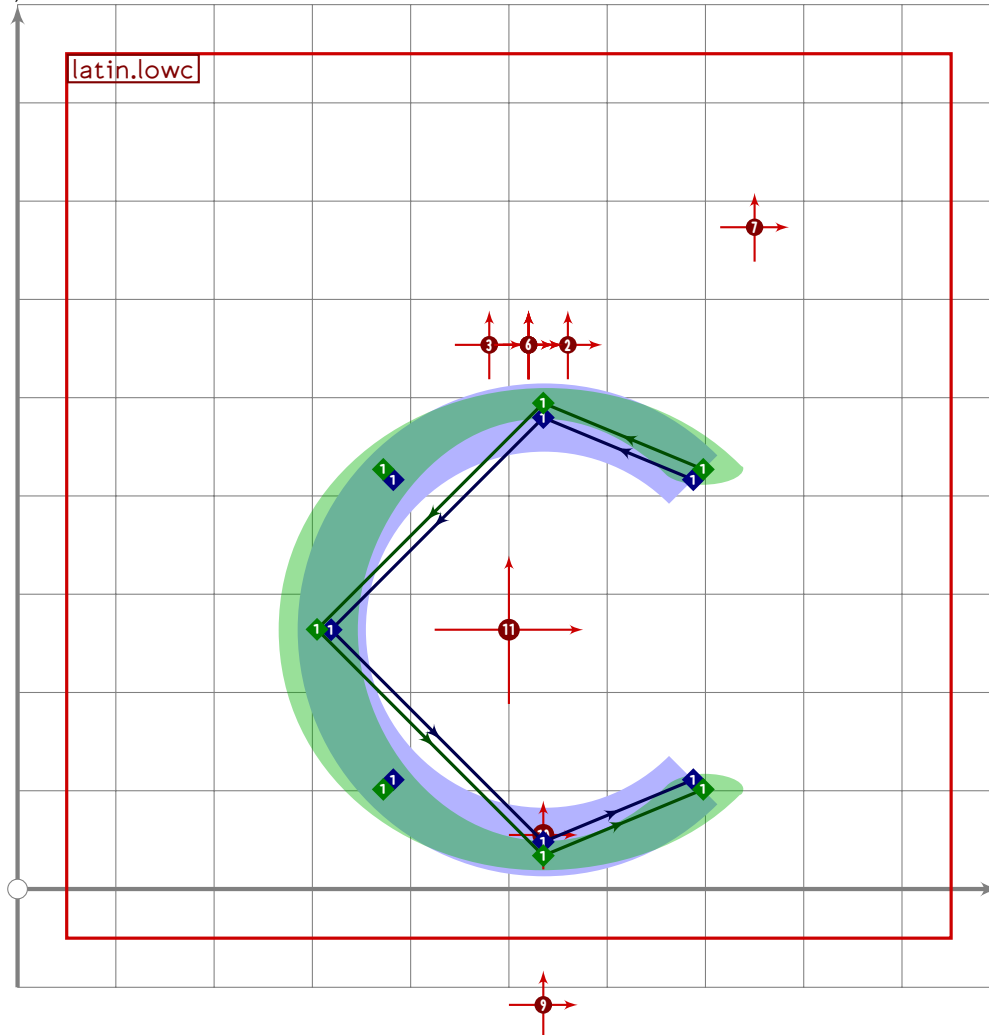
1011
1012 vardef latin.lowb =
1013   push_pbox_toexpand("latin.lowb");
1014   (x1+x4)/2=500;
1015   (x4-x1)=(y1-y3)*0.55;
1016   x2=x1=x6;
1017   x3=0.4[x2,x4];
1018   x5=0.55[x2,x4];
1019
1020   y1=latin_wide_high_v;
1021   y2=latin_wide_lc_baselift;
1022   y3=latin_wide_low_r;
1023   y4=0.48[y3,y5];
1024   y5=latin_wide_xheight_h;
1025   y6=0.77[y3,y5];
1026
1027   push_stroke(z1-z2{curl 0.05}{right}z3.{up}z4.{left}z5..z6,
1028     (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
1029   replace_strokep(0)(subpath (0,4.97) of oldp);
1030   set_botip(0,1,1);
1031   if not do_italic_hook: set_boserif(0,0,1); fi;

```

```

1032
1033 push_anchor(anc_wide,identity xscaled 0.9
1034   transformed accent_default[anc_wide] shifted (30,10));
1035 push_anchor(anc_tilde,identity xscaled 0.8
1036   transformed accent_default[anc_tilde] shifted (30,10));
1037 push_anchor(anc_ring,accent_default[anc_ring] shifted (-10,-10));
1038 expand_pbox;
1039 enddef;

```



```

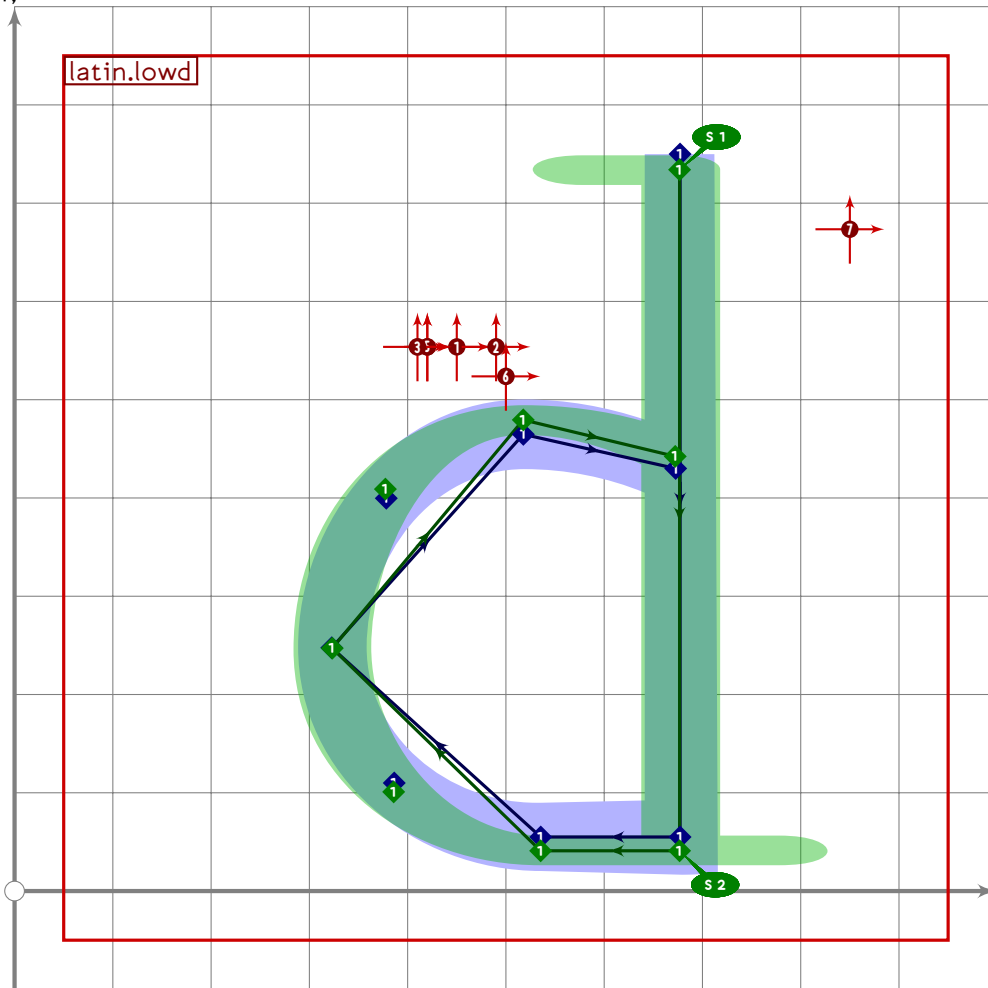
1040
1041 vardef latin.lowc =
1042   push_pbox_toexpand("latin.lowc");
1043   push_stroke((subpath (0,5,3,5) of ((1,0)..(0,1)..(-1,0)..(0,-1)..cycle))
1044     scaled ((latin_wide_xheight_r-latin_wide_low_r)/2)
1045     shifted ((xpart centre_pt,(latin_wide_xheight_r+latin_wide_low_r)/2)
1046       +(35,0)),
1047     (1,6,1,6)-(1,6,1,6)-(1,6,1,6)-(1,6,1,6)-(1,6,1,6));
1048   tsu_accent.shift_anchors(ypart olda>vmetric(0.52))((20,0));
1049   tsu_accent.shift_anchors(ypart olda<=vmetric(0.52))((35,0));
1050   push_anchor(anc_centre,identity
1051     scaled ((latin_wide_xheight_r-latin_wide_low_r)/200)

```

```

1052   shifted (xpart centre_pt,(latin_wide_xheight_r+latin_wide_low_r)/2));
1053   expand_pbox;
1054 enddef;

```



```

1055
1056 vardef latin.lowd =
1057   push_pbox_toexpand("latin.lowd");
1058   (x1+x4)/2=500;
1059   (x1-x4)=(y1-y3)*0.51;
1060   x2=x1=x6;
1061   x3=0.4[x2,x4];
1062   x5=0.45[x2,x4];
1063
1064   y1=latin_wide_high_v;
1065   y2=y3=latin_wide_low_h;
1066   y4=0.47[y3,y5];
1067   y5=latin_wide_xheight_h;
1068   y6=0.91[y3,y5];
1069
1070   push_stroke(z1-z2{left}..{left}z3.{up}z4.{right}z5..z6,
1071     (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-
1072     (1.6,1.6)-(1.6,1.6));

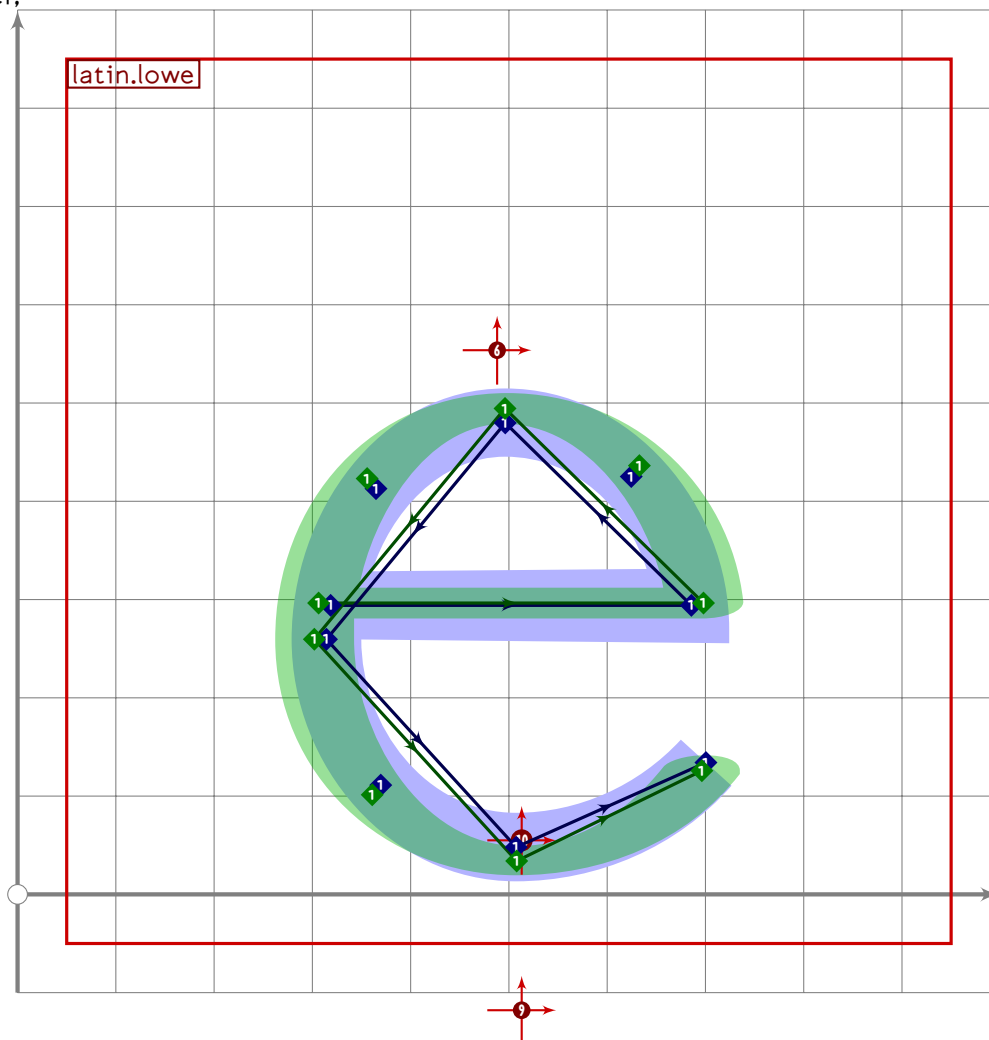
```

U+FF45  
tsuku.uniFF45

```

1073 replace_strokep(0)(subpath (0,4.97) of oldp);
1074 set_botip(0,1,1);
1075 if not do_italic_hook: set_boserif(0,0,1); fi;
1076 set_boserif(0,1,2);
1077
1078 push_anchor(anc_wide,identity xscaled 0.9
1079   transformed accent_default[anc_wide] shifted (-30,0));
1080 push_anchor(anc_tilde,identity xscaled 0.8
1081   transformed accent_default[anc_tilde] shifted (-30,0));
1082
1083 tsu_accent.shift_anchors(ypart olda>vmetric(0.52))((-50,0));
1084 push_anchor(anc_ring,accent_default[anc_ring] shifted (0,-30));
1085 push_anchor(anc_caron_comma,accent_default[anc_caron_comma] shifted (120,0));
1086 expand_pbox;
1087 enddef;

```



LATI

```

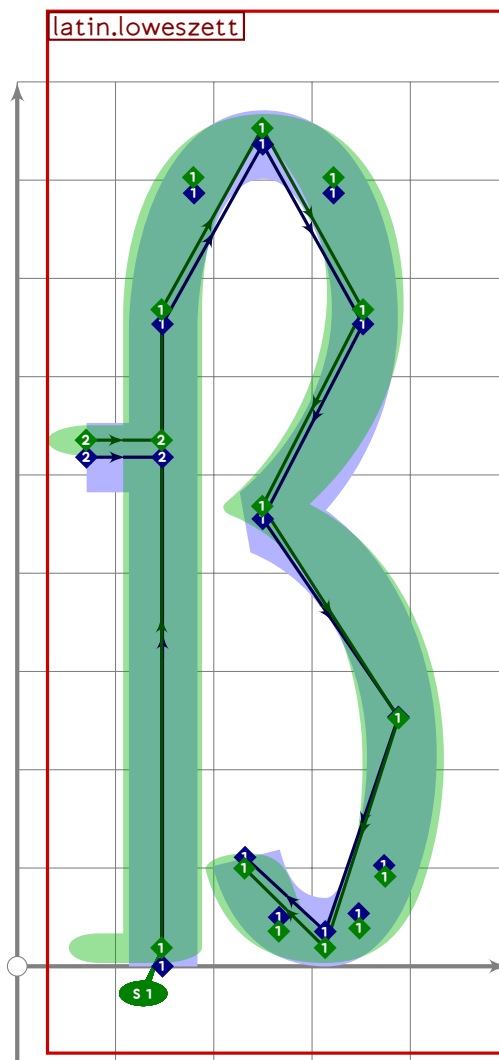
1088
1089 vardef latin.lowe =
1090   push_pbox_toexpand("latin.lowe");
1091   y2=0.57[y5,y3];
1092   y3=latin_wide_xheight_r;

```

```

1093 y4=0.49[y5,y3];
1094 y5=latin_wide_low_r;
1095 y6=0.35[y5,y2];
1096
1097 (x2+x4)/2=500;
1098 (x2-x4)=0.86*(y3-y5);
1099 x3=0.49[x4,x2];
1100 x5=0.52[x4,x2];
1101 x6=1.04[x4,x2]-(if sharp_corners: 0 else: (mbrush_width/3) fi);
1102
1103 push_stroke(z2{curl 0.7}..z3{left}..z4{down}..z5{right}..z6,
1104   (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
1105 z1=get_strokep(0) intersectionpoint ((z2+(-1000,0))-z2+(-10,0));
1106 replace_strokep(0)((z1+2*right)-oldp);
1107 set_botip(0,1,1);
1108
1109 tsu_accent.shift_anchors(ypart olda<vmetric(0.05))((13,0));
1110 tsu_accent.shift_anchors(ai=anc_ring)((-12,0));
1111 expand_pbox;
1112 enddef;

```



```

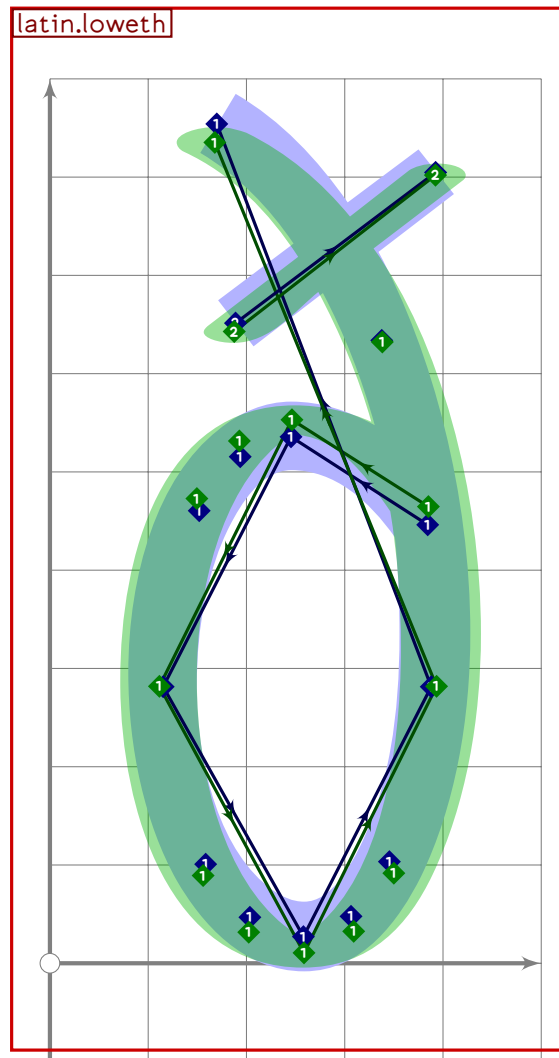
1113
1114 vardef latin.loweszett =
1115   push_pbox_toexpand("latin.loweszett");
1116   (x1+x6)/2=510;
1117   (x6-x1)=3*(y3-y2);
1118   x2=x1;
1119   x3=x5=(x1+x4)/2;
1120   x4=0.85[x1,x6];
1121   x7=0.69[x1,x6];
1122   x8=0.35[x1,x6];
1123
1124   y1=latin_wide_low_v;
1125   y2=y4=0.52[y5,y3];
1126   y3=latin_wide_high_r;
1127   y5=0.87[y7,latin_wide_xheight_h];
1128   y6=0.52[y7,y5];
1129   y7=latin_wide_low_h;
1130   y8=0.18[y7,y5];
1131

```

```

1132 push_stroke(z1-z2{dir 88}..z3{right}..z4.{dir 200}z5{dir 350}..
1133     z6..z7{left}..z8{dir 120},
1134     (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-
1135     (1.6,1.6)-(1.6,1.6)-(1.6,1.6));
1136 set_botip(0,4,0);
1137 set_boserif(0,0,1);
1138
1139 push_stroke((x1-150,latin_wide_xheight_h)-(x1,latin_wide_xheight_h),
1140     (1.6,1.6)-(1.6,1.6));
1141 expand_pbox;
1142 enddef;

```



```

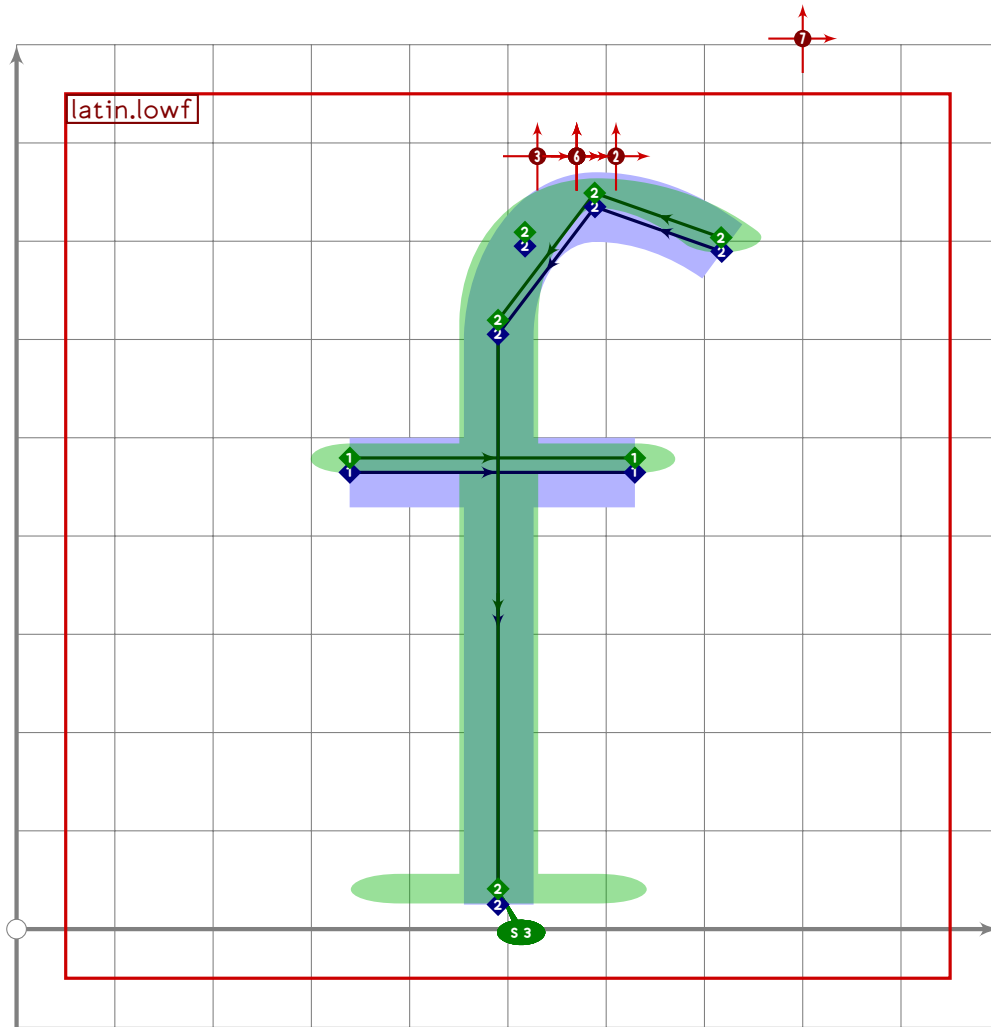
1143
1144 vardef latin.loweth =
1145     push_pbox_toexpand("latin.loweth");
1146     (x3+x5)/2=(x2+x4)/2=510;
1147     x4-x2=20;
1148     (x5-x3)=(y2-y4);
1149     x1=1.3[x3,x5];
1150     x6=0.2[x3,x5];

```

```

1151
1152 y1=0.25[y4,y2];
1153 y3=y5=0.5[y4,y2];
1154 y2=latin_wide_xheight_r;
1155 y4=latin_wide_low_r;
1156 y6=latin_wide_high_v;
1157
1158 push_stroke(z1..z2{left}..z3..z4{right}..z5.{curl 0.6}z6,
1159   (1.6,1.6)–(1.6,1.6)–(1.6,1.6)–(1.6,1.6)–(1.6,1.6)–(1.6,1.6));
1160 replace_strokep(0)(subpath (xpart ((subpath (0,1) of oldp)
1161   intersectiontimes
1162     (subpath (2,infinity) of oldp)),infinity) of oldp);
1163
1164 z7=point 4.67 of get_strokep(0);
1165 z8=z7+whatever*dir 202;
1166 x8=0.27[x3,x5];
1167 z9=2[z8,z7];
1168
1169 push_stroke(z8–z9,(1.5,1.5)–(1.5,1.5));
1170 set_bosize(0)(85);
1171 expand_pbox;
1172 enddef;

```



```

1173
1174 vardef latin.lowf =
1175   push_pbox_toexpand("latin.lowf");
1176   (x2-x1)=290;
1177   x5=x6=490=0.52[x1,x2];
1178   x3-x5=2*(y4-y5);
1179   x4=0.38[x5,x3];
1180
1181   y1=y2=latin_wide_xheight_h;
1182   y5=0.52[y2,y4];
1183   y3=0.73[y2,y4];
1184   y4=latin_wide_high_r;
1185   y6=latin_wide_low_v;
1186
1187   push_stroke(z1-z2,(1.6,1.6)-(1.6,1.6));
1188
1189   push_stroke(z3{curl 0.6}..z4{left}..{dir 268}z5{down}-z6,
1190     (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
1191   replace_strokep(0)(subpath (0.23,3) of oldp);
1192   set_boserif(0,3,3);

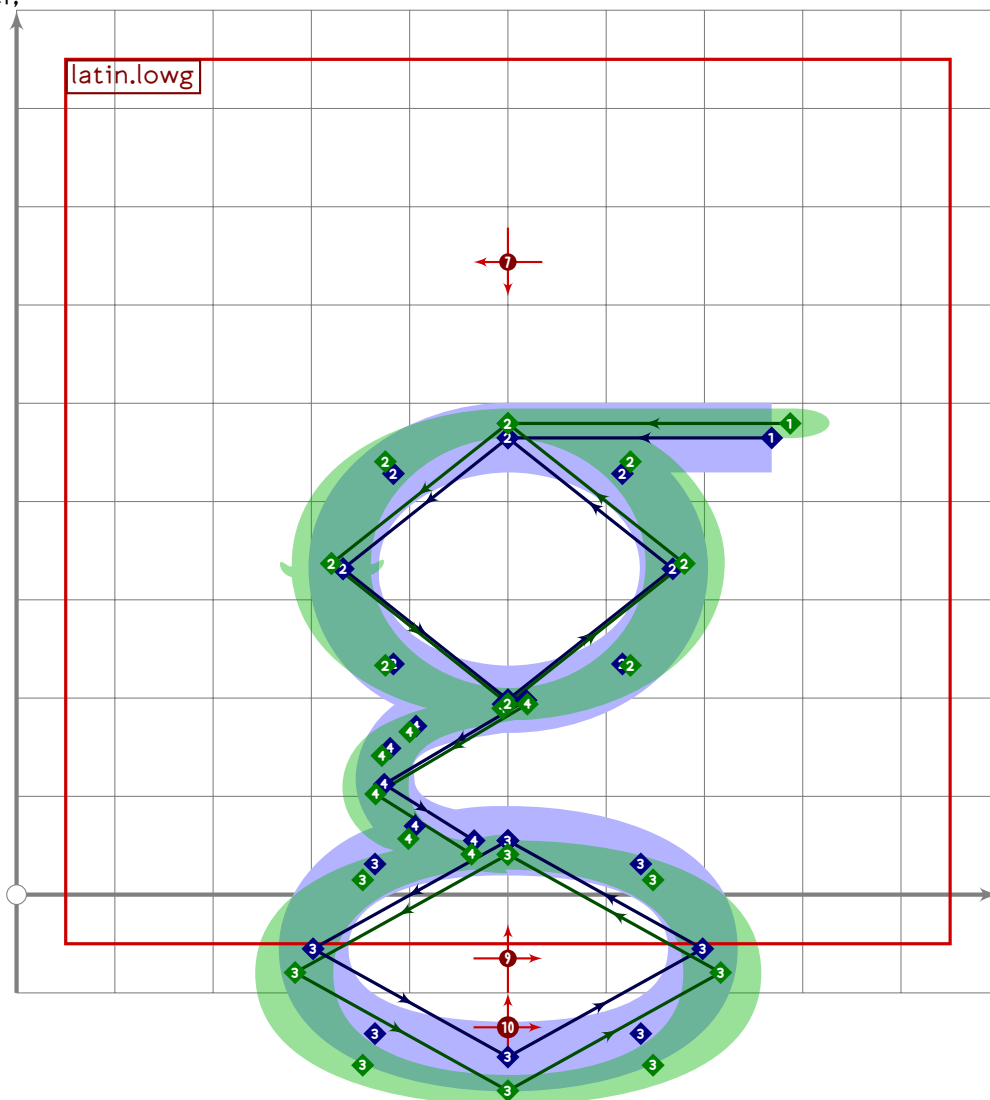
```

U+FF47  
tsuku.uniFF47

```

1193
1194 tsu_accent.shift_anchors(ypart olda>vmetric(0.52))
1195 (((0,0) transformed tsu_xf.cap_upper_accent)-
1196 ((0,0) transformed accent_default[anc_upper])+(70,0));
1197 expand_pbox;
1198 enddef;

```



```

1199
1200 vardef latin.lowg =
1201   push_pbox_toexpand("latin.lowg");
1202   x2=x4=x7=x9=500;
1203   x1=1.6[x2,x5];
1204   x5-x2=x2-x3;
1205   x5-x3=1.26*(y2-y4);
1206   x6=0.25[x3,x2];
1207   x10-x7=x7-x8;
1208   x10-x8=1.8*(y7-y9);
1209
1210   y1=y2=latin_wide_xheight_h;

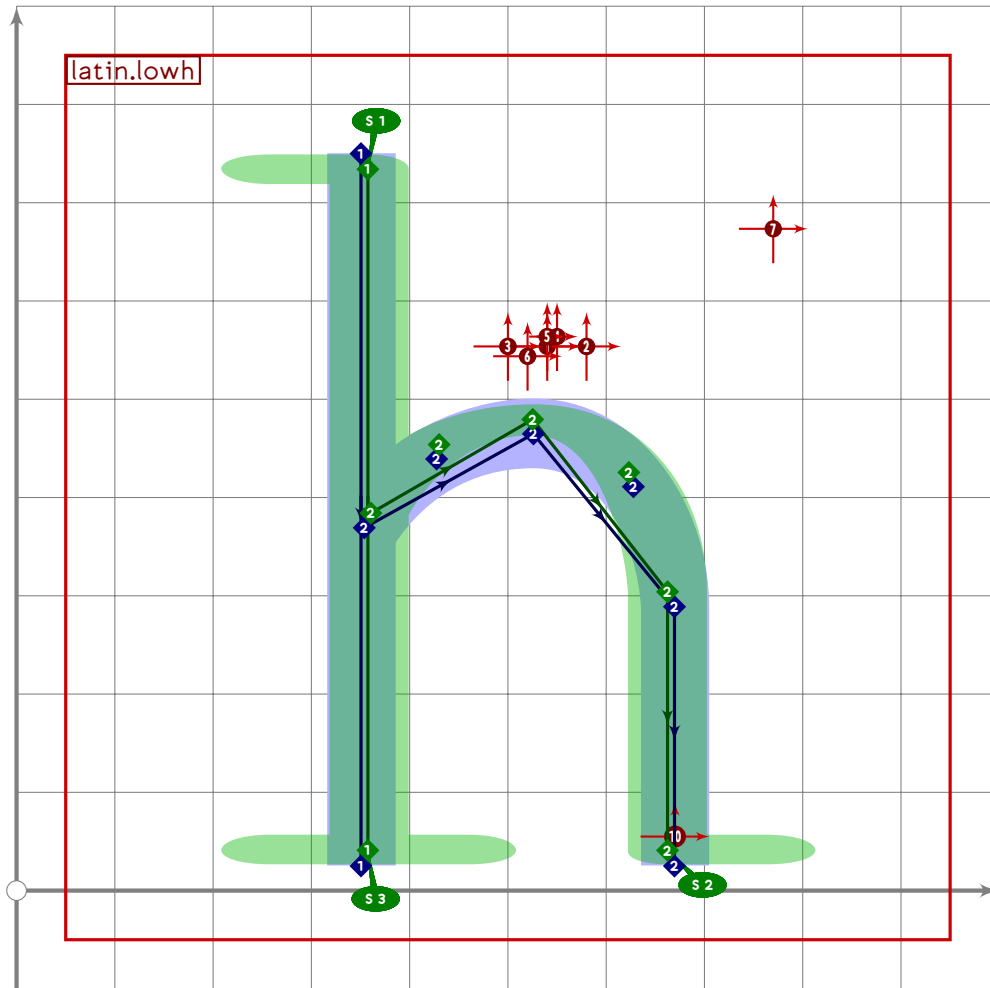
```

**LATI**

```

1211 y3=y5=0.5[y4,y2];
1212 y4=0.35[y7,y2];
1213 y6=0.4[y7,y4];
1214 y7=latin_wide_low_h;
1215 y8=y10=0.5[y9,y7];
1216 y9=latin_wide_desc_r;
1217
1218 push_stroke(z1-z2,(1.6,1.6)-(1.6,1.6));
1219
1220 push_stroke(z3..z4..z5..z2..cycle,
1221   (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-cycle);
1222
1223 push_stroke(z7..z8..z9..z10..cycle,
1224   (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-cycle);
1225
1226 push_stroke((point 1.2 of get_stroke(-1))
1227   {-direction 1.2 of get_stroke(-1)}..z6..
1228   (point 0.05 of get_stroke(0)) {-direction 0.05 of get_stroke(0)},
1229   (1.6,1.6)-(1.6,1.6)-(1.6,1.6));
1230 replace_stroke(0)(subpath (0.13,1.87) of oldp);
1231 set_bosize(0,85);
1232
1233 push_anchor(anc_caron_comma,
1234   identity rotated 180 shifted (0,90)
1235   transformed accent_default[anc_upper]);
1236 push_anchor(anc_lower,
1237   accent_default[anc_lower] shifted (0,53));
1238 push_anchor(anc_lower_connect,
1239   accent_default[anc_lower_connect] shifted (0,-190));
1240 expand_pbox;
1241 endif;

```



```

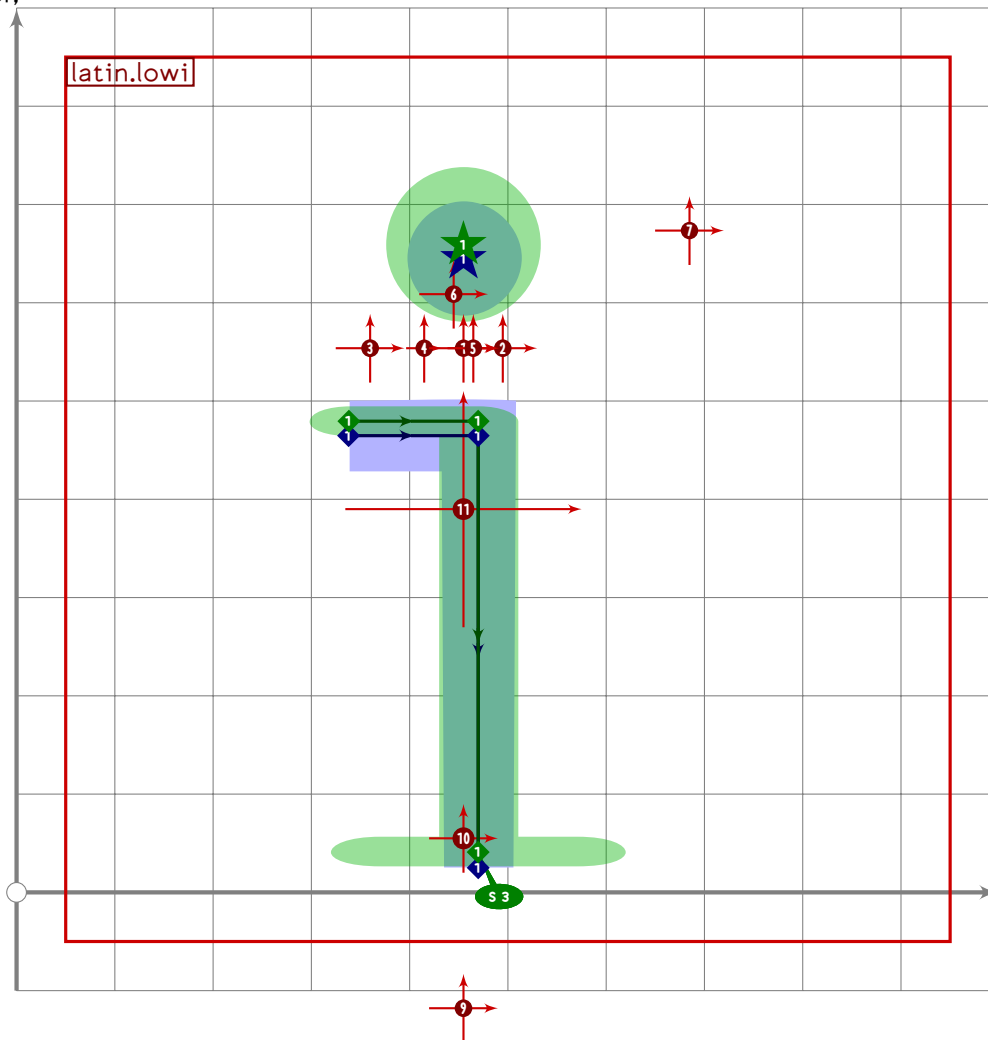
1242
1243 vardef latin.lowh =
1244   push_pbox_toexpand("latin.lowh");
1245   (x1+x5)/2=510;
1246   (x5-x1)=(y1-y2)*0.44;
1247   x2=x1=x3;
1248   x4=0.55[x3,x5];
1249   x6=x5;
1250
1251   y1=latin_wide_high_v;
1252   y2=y6=latin_wide_low_v;
1253   y3=0.77[y2,y4];
1254   y4=latin_wide_xheight_h;
1255   y5=0.60[y2,y4];
1256
1257   push_stroke(z1-z2,(1.6,1.6)-(1.6,1.6));
1258   if not do_italic_hook:
1259     set_boserif(0,0,1);
1260     set_boserif(0,1,3);
1261   fi;
1262

```

```

1263 push_stroke(z3..z4{right}..z5{dir 273}-z6,
1264   (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
1265 replace_stroke(0)(subpath (0.03,3) of oldp);
1266 set_boserif(0,3;if do_italic_hook: 11 else: 2 fi);
1267
1268 tsu_accent.shift_anchors(ypart olda>vmetric(0.52))((40,0));
1269 push_anchor(anc_wide,identity xscaled 0.8
1270   transformed accent_default[anc_wide] shifted (50,10));
1271 push_anchor(anc_tilde,identity xscaled 0.7
1272   transformed accent_default[anc_tilde] shifted (40,10));
1273 push_anchor(anc_ring,accent_default[anc_ring] shifted (20,-10));
1274 push_anchor(anc_lower_connect,
1275   accent_default[anc_lower_connect] shifted (170,0));
1276 expand_pbox;
1277 enddef;

```



```

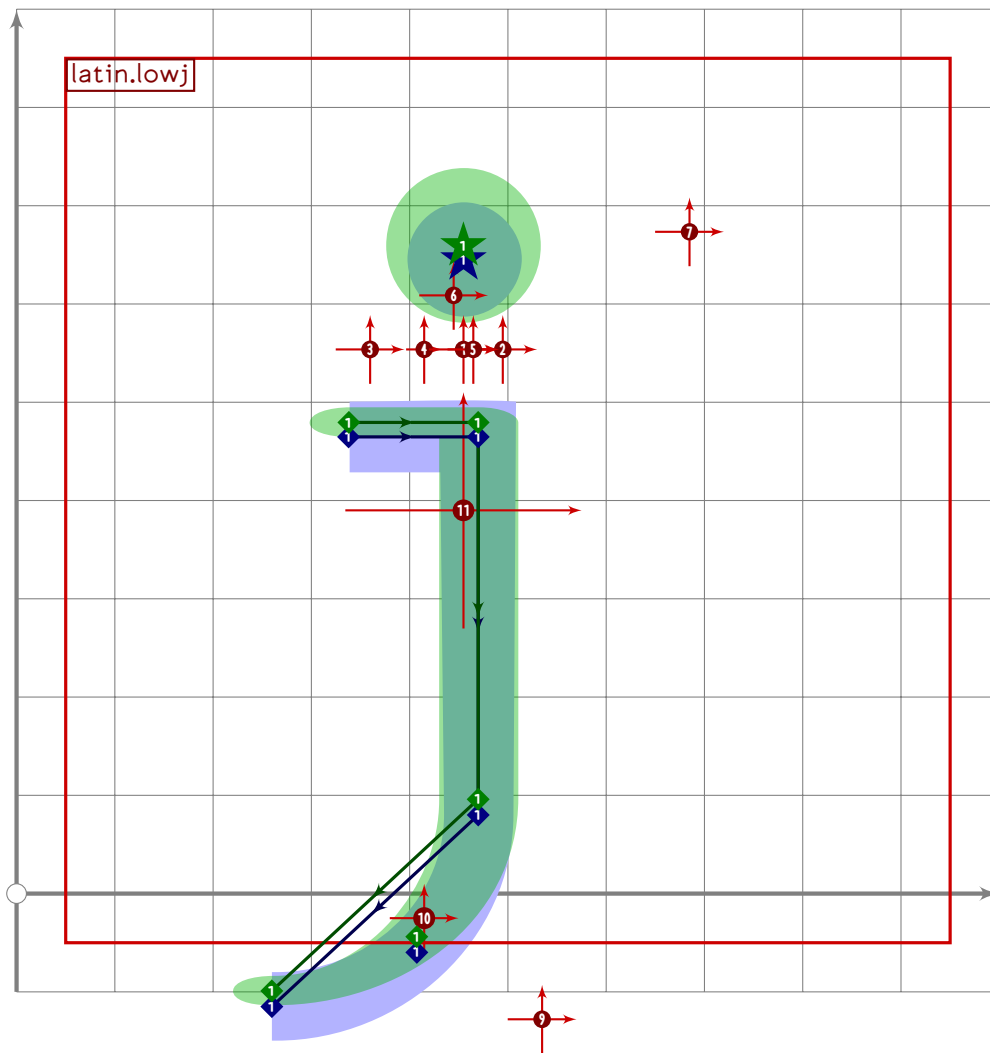
1278
1279 vardef latin.lowi =
1280   push_pbox_toexpand("latin.lowi");
1281   x2=x3;
1282   0.85[x1,x2]=450;

```

```

1283 (x2-x1)=0.3(y2-y3);
1284 x4=x2-15;
1285
1286 y1=y2=latin_wide_xheight_h;
1287 y3=latin_wide_low_v;
1288 y4=0.5[y2,latin_wide_high_v]+mbrush_width;
1289
1290 push_stroke(z1-z2-z3,(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
1291 set_botip(0,1,1);
1292 set_boserif(0,2,3);
1293
1294 push_lcblob(fullcircle rotated 45 scaled (mbrush_width*2.7+15)
1295   shifted (z4 transformed tsu_rescale_xform)
1296   transformed inverse tsu_rescale_xform);
1297
1298 push_anchor(anc_wide,
1299   identity xscaled 0.7 transformed accent_default[anc_wide]);
1300 tsu_accent.shift_anchors(true)((x4-500,0));
1301 tsu_accent.shift_anchors(ai=anc_acute)((-55,0));
1302 tsu_accent.shift_anchors(ai=anc_wide)((-40,0));
1303 tsu_accent.shift_anchors(ai=anc_tilde)((10,0));
1304 tsu_accent.shift_anchors(ai=anc_ring)((-10,55));
1305 expand_pbox;
1306 endif;

```



```

1307
1308 vardef latin.lowj =
1309   push_pbox_toexpand("latin.lowj");
1310   x2=x3;
1311   0.85[x1,x2]=450;
1312   (x2-x1)=0.3(y2-y3);
1313   x5=x2-15;
1314
1315   y1=y2=latin_wide_xheight_h;
1316   y3=latin_wide_low_v;
1317   y5=0.5[y2,latin_wide_high_v]+mbrush_width;
1318
1319   z4=z3+(-210,-140);
1320
1321   push_stroke((z1-z2-(z3+(0,55)))..{curl 0.8}z4,
1322     (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
1323   set_botip(0,1,1);
1324
1325   push_lcblob(fullcircle scaled (mbrush_width*2.7*15)

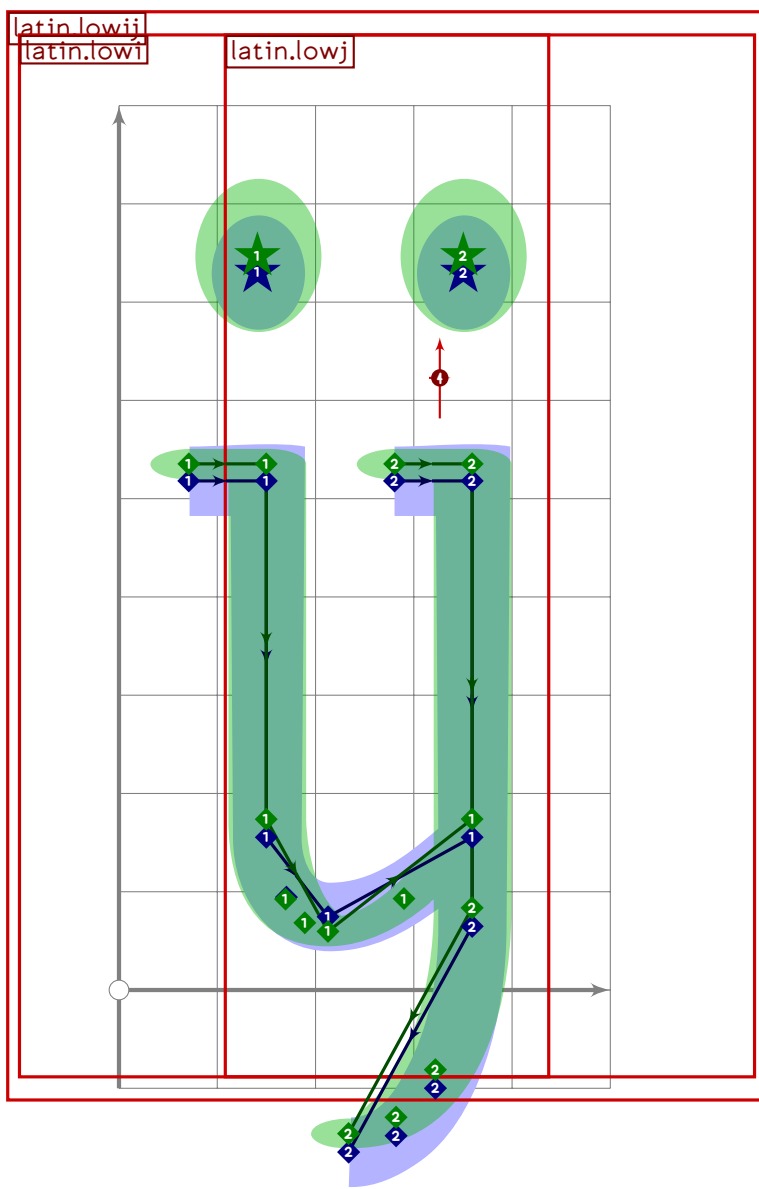
```

U+0133  
tsuku.ij

```

1326 shifted (z5 transformed tsu_rescale_xform)
1327 transformed inverse tsu_rescale_xform);
1328
1329 push_anchor(anc_wide,
1330 identity xscaled 0.7 transformed accent_default[anc_wide]);
1331 tsu_accent.shift_anchors(true)((x5-500,0));
1332 tsu_accent.shift_anchors(ai=anc_acute)((-55,0));
1333 tsu_accent.shift_anchors(ai=anc_wide)((-40,0));
1334 tsu_accent.shift_anchors(ai=anc_tilde)((10,0));
1335 tsu_accent.shift_anchors(ai=anc_ring)((-10,55));
1336 tsu_accent.shift_anchors(ai=anc_lower)((80,-10));
1337 tsu_accent.shift_anchors(ai=anc_lower_connect)((-40,-80));
1338 expand_pbox;
1339 endif;

```



LATI

```

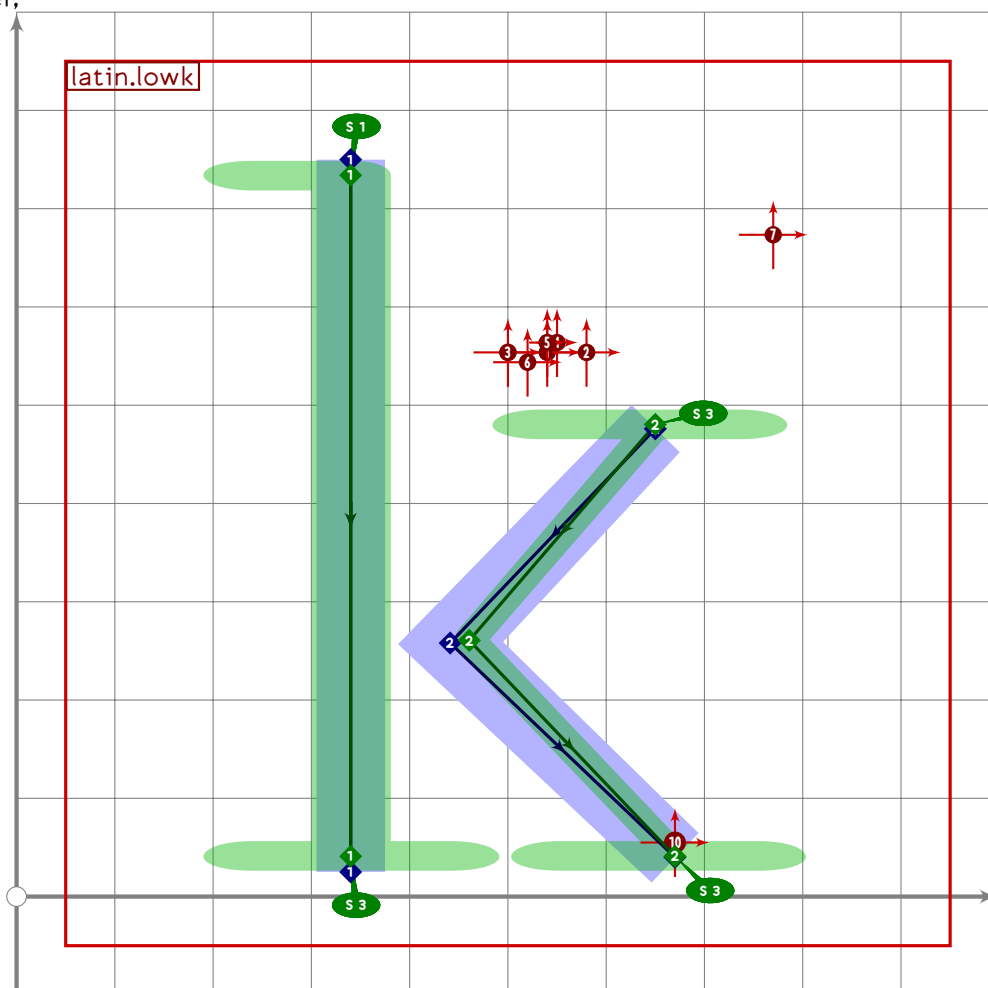
1340
1341 vardef latin.lowij =

```

```

1342 push_pbox_toexpand("latin.lowij");
1343 tsu_xform(identity shifted (-200,0))(latin.lowi);
1344 numeric x[],y[];
1345 tsu_xform(identity shifted (150,0))(latin.lowj);
1346 replace_lcblob(-1)(get_lcblob(0) shifted (-350,0));
1347 numeric x[],y[];
1348 z1=point 1.7 of get_strokep(-1);
1349 x2=0.3[x1,x3];
1350 y2=vmetric(0.05);
1351 z3=point 1.8 of get_strokep(0);
1352 replace_strokep(-1)((subpath (0,1) of oldp)-
1353   z1{dir 273}..z2{right}..{curl 0.3}z3);
1354 replace_strokeq(-1)((subpath (0,1) of oldq)-(1.6,1.6)-(1.6,1.6)-(1,1));
1355 set_boserif(-1,2,whatever);
1356 expand_pbox;
1357 enddef;

```



```

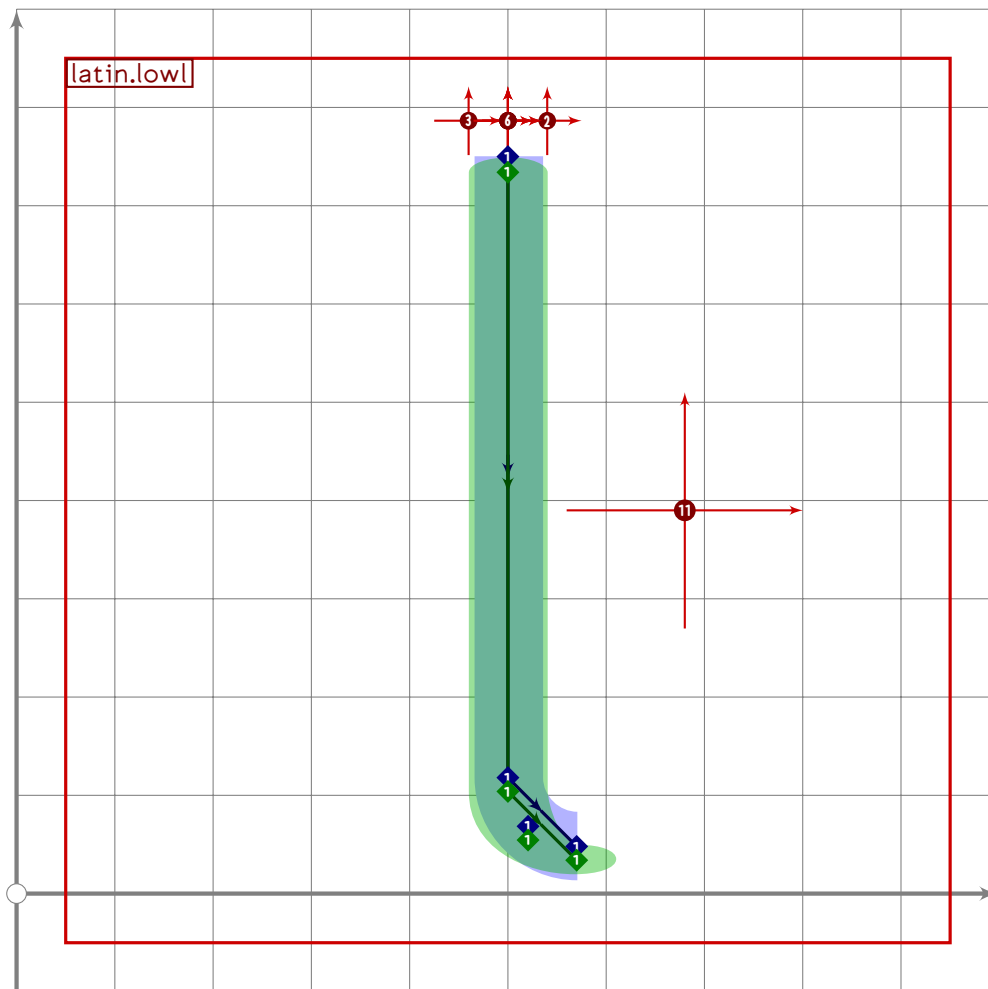
1358
1359 vardef latin.lowk =
1360   push_pbox_toexpand("latin.lowk");
1361   z1=(340,latin_wide_high_v);
1362   z2=(340,latin_wide_low_v);

```

```

1363 z3=(650,(latin_wide_xheight_v+latin_wide_xheight_h)/2);
1364 x4=340+mbrush_width*if sharp_corners: 2.7 else: 2.3 fi;
1365 y4=(y3+y5)/2;
1366 z5=(670,0.5[latin_wide_low_h,latin_wide_low_v]);
1367
1368 push_stroke(z1-z2,(1.6,1.6)-(1.6,1.6));
1369 if not do_italic_hook:
1370     set_boserif(0,0,1);
1371     set_boserif(0,1,3);
1372 fi;
1373
1374 push_stroke(z3-z4-z5,(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
1375 set_botip(0,1,1);
1376 set_boserif(0,0,if do_italic_hook: 1 else: 3 fi);
1377 if not do_italic_hook: set_boserif(0,2,3); fi;
1378 set_boaltnate(0);
1379
1380 tsu_accent.shift_anchors(y part olda>vmetric(0.52))((40,0));
1381 push_anchor(anc_wide,identity xscaled 0.8
1382     transformed accent_default[anc_wide] shifted (50,10));
1383 push_anchor(anc_tilde,identity xscaled 0.7
1384     transformed accent_default[anc_tilde] shifted (40,10));
1385 push_anchor(anc_ring,accent_default[anc_ring] shifted (20,-10));
1386 push_anchor(anc_lower_connect,
1387     accent_default[anc_lower_connect] shifted (170,0));
1388 expand_pbox;
1389 enddef;

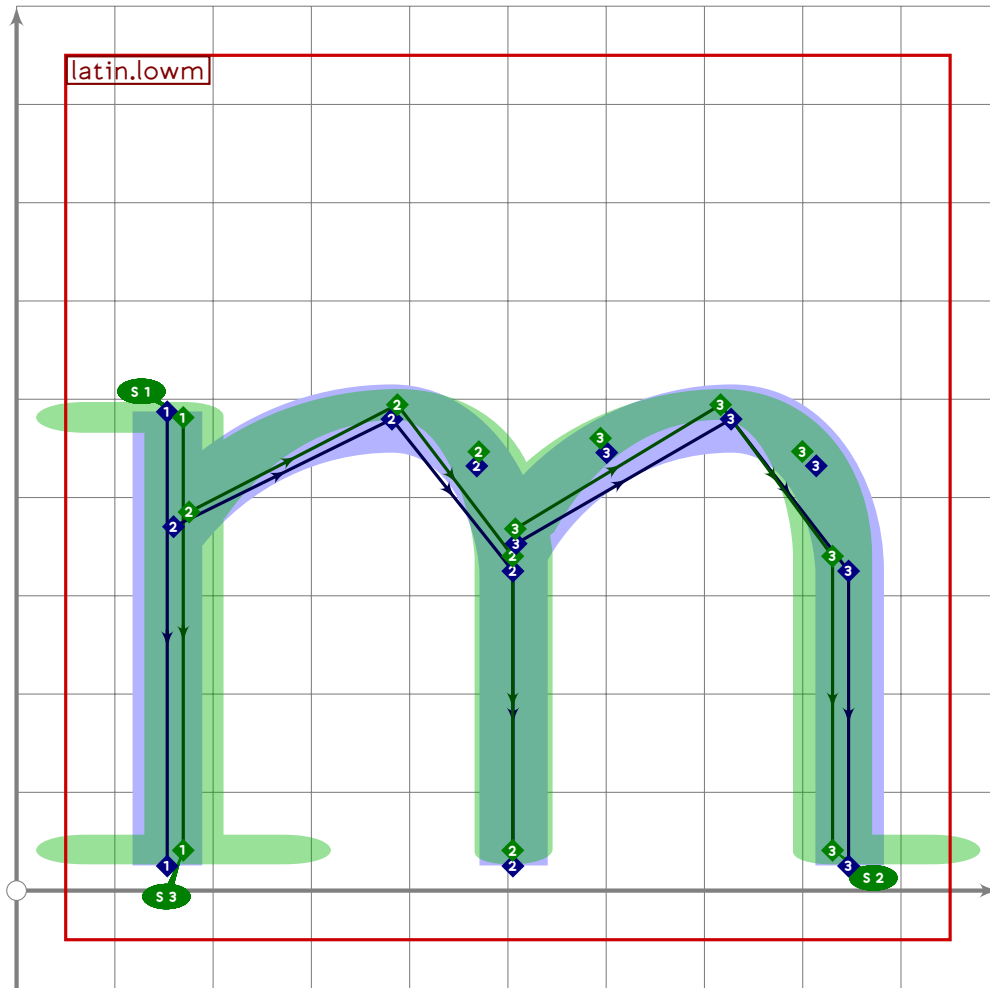
```



```

1390
1391 vardef latin.lowl =
1392   push_pbox_toexpand("latin.lowl");
1393   x1=x2=500;
1394   x3=570;
1395
1396   y1=latin_wide_high_v;
1397   y2=y3+(x3-x2);
1398   y3=latin_wide_low_r;
1399
1400   push_stroke(z1-z2{down}..{right}z3,(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
1401
1402   tsu_accent.shift_anchors((ypart olda>vmetric(0.52))
1403                           and not (ai=anc_caron_comma))
1404   (((0,0) transformed tsu_xf.cap_upper_accent)-
1405    ((0,0) transformed accent_default[anc_upper]));
1406   tsu_accent.shift_anchors(ai=anc_centre)((180,0));
1407   expand_pbox;
1408 enddef;

```



```

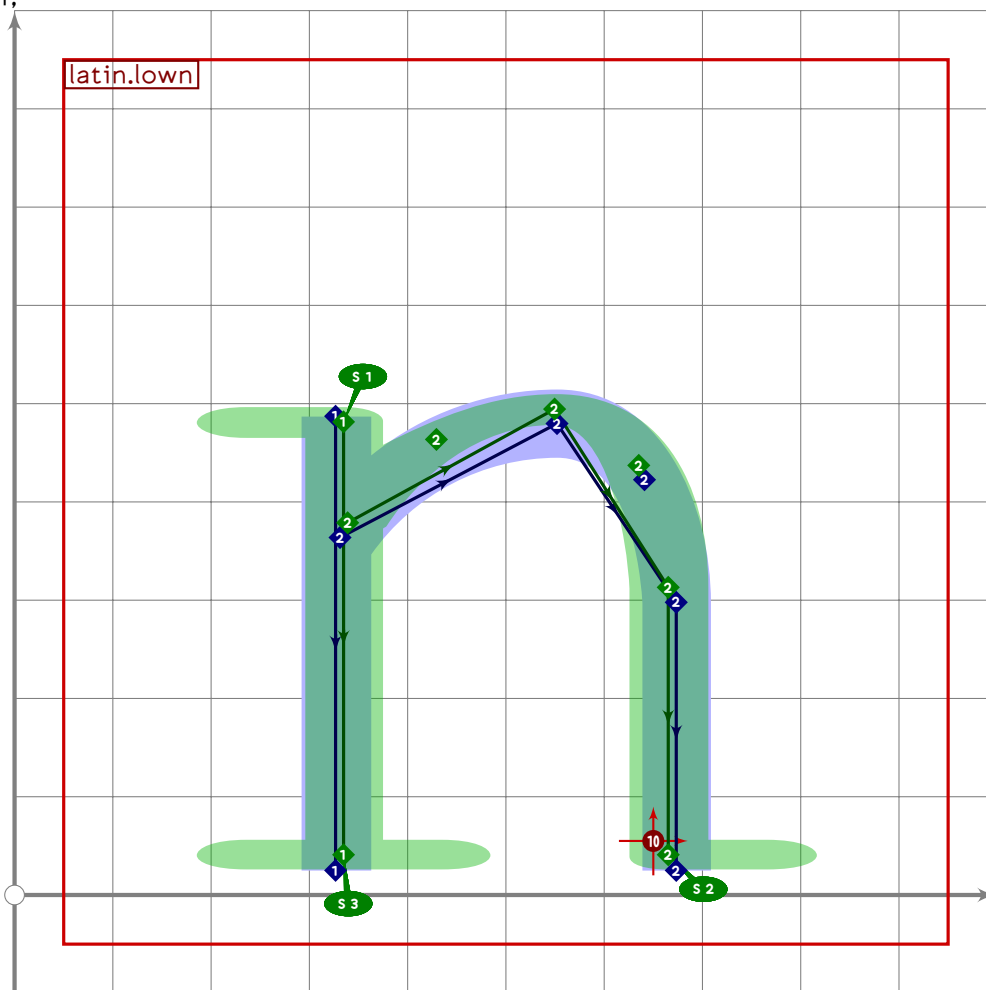
1409
1410 vardef latin.lowm =
1411   push_pbox_toexpand("latin.lowm");
1412   (x1+x9)/2=500;
1413   (x9-x1)*2=(y1-y2)*3;
1414   (x5-x1)=(x9-x5)*1.03;
1415   x2=x1=x3;
1416   x4=0.65[x3,x5];
1417   x6=x5;
1418   x8=0.65[x6,x9];
1419   x9=x10;
1420
1421   y1=latin_wide_xheight_v;
1422   y2=y6=y10=latin_wide_low_v;
1423   y3=0.74[y2,y4];
1424   y4=y8=latin_wide_xheight_r;
1425   y5=y9=0.66[y2,y4];
1426
1427   push_stroke(z1-z2,(1.6,1.6)-(1.6,1.6));
1428   set_boserif(0,0;if do_italic_hook: 11 else: 1 fi);
1429   if not do_italic_hook: set_boserif(0,1,3); fi;

```

```

1430
1431 push_stroke(z3..z4{right}..z5{dir 275}-z6,
1432   (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
1433 replace_strokep(0)(subpath (0.04,3) of oldp);
1434
1435 z7=get_strokep(0) intersectionpoint (z3-z9);
1436
1437 push_stroke(z7..z8{right}..z9{dir 271}-z10,
1438   (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
1439 replace_strokep(0)(subpath (0.04,3) of oldp);
1440 set_boserif(0,3;if do_italic_hook: 11 else: 2 fi);
1441 expand_pbox;
1442 enddef;

```



```

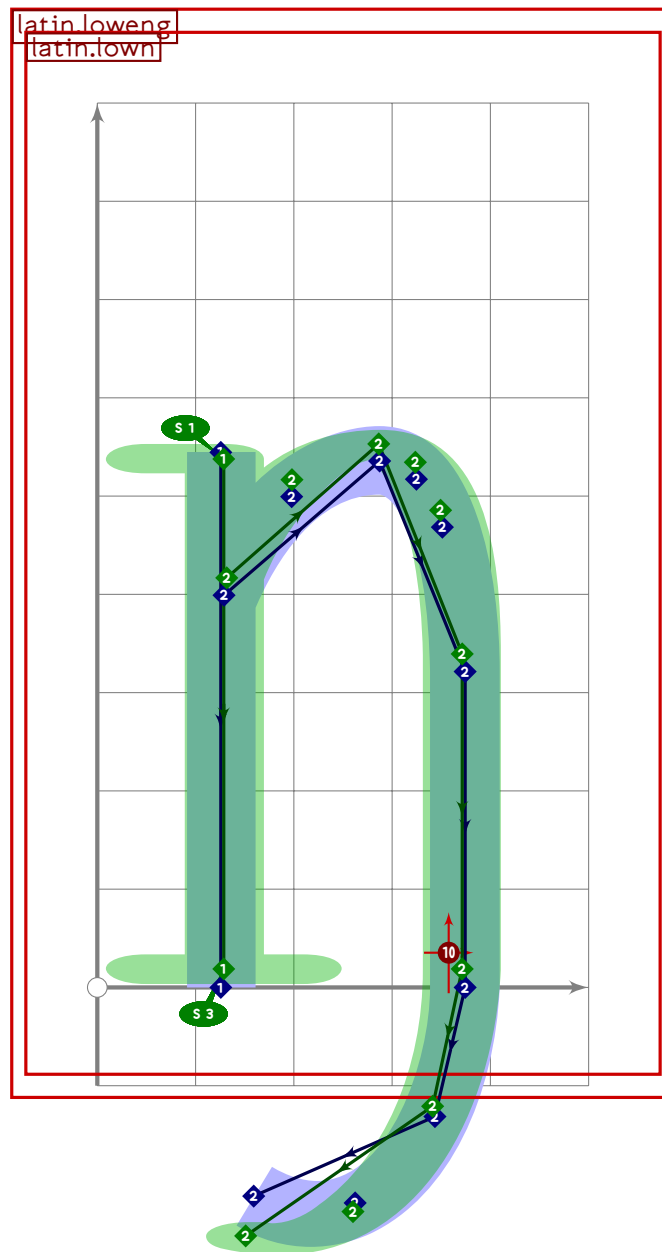
1443
1444 vardef latin.lown =
1445   push_pbox_toexpand("latin.lown");
1446   (x1+x5)/2=500;
1447   (x5-x1)=(y1-y2)*0.75;
1448   x2=x1=x3;
1449   x4=0.65[x3,x5];
1450   x6=x5;

```

```

1451
1452 y1=latin_wide_xheight_v;
1453 y2=y6=latin_wide_low_v;
1454 y3=0.73[y2,y4];
1455 y4=latin_wide_xheight_r;
1456 y5=0.60[y2,y4];
1457
1458 push_stroke(z1-z2,(1.6,1.6)-(1.6,1.6));
1459 set_boserif(0,0;if do_italic_hook: 11 else: 1 fi);
1460 if not do_italic_hook: set_boserif(0,1,3); fi;
1461
1462 push_stroke(z3..z4{right}..z5{dir 273}-z6,
1463   (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
1464 replace_strokep(0)(subpath (0.03,3) of oldp);
1465 set_boserif(0,3;if do_italic_hook: 11 else: 2 fi);
1466
1467 push_anchor(anc_lower_connect,
1468   accent_default[anc_lower_connect] shifted (150,0));
1469 expand_pbox;
1470 enddef;

```

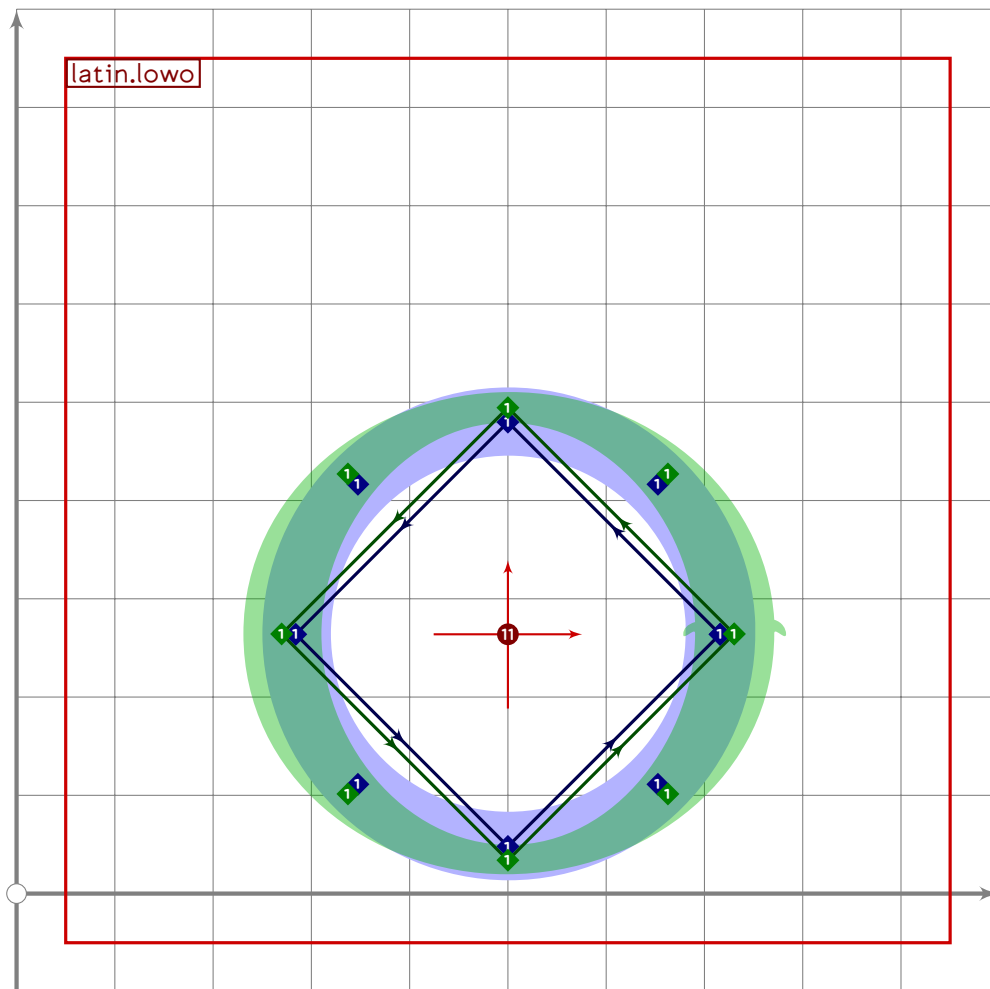


```

1471
1472 vardef latin.loweng =
1473   push_pbox_toexpand("latin.loweng");
1474   latin.lown;
1475   x7=x6-300;
1476   y7=latin_wide_desc_h;
1477   replace_strokep(0)(oldp{dir 266}..{curl 0.8}z7);
1478   replace_strokep(0)(insert_nodes(oldp)(3.3));
1479   replace_strokeq(0)(oldq-(1.6,1.6)-(1.6,1.6));
1480   set_boserif(0,3,whatever);
1481   expand_pbox;
1482 enddef;

```

U+FF4F  
tsuku.uniFF4F

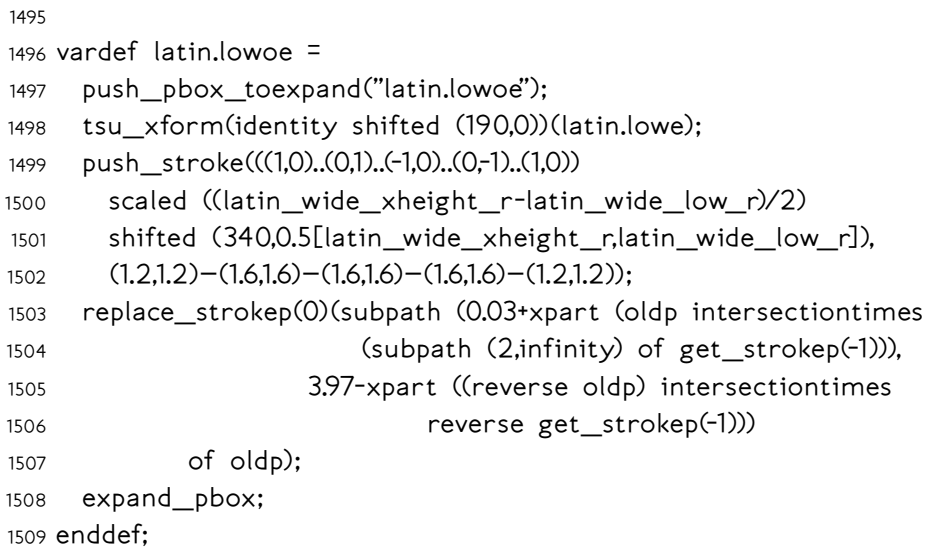


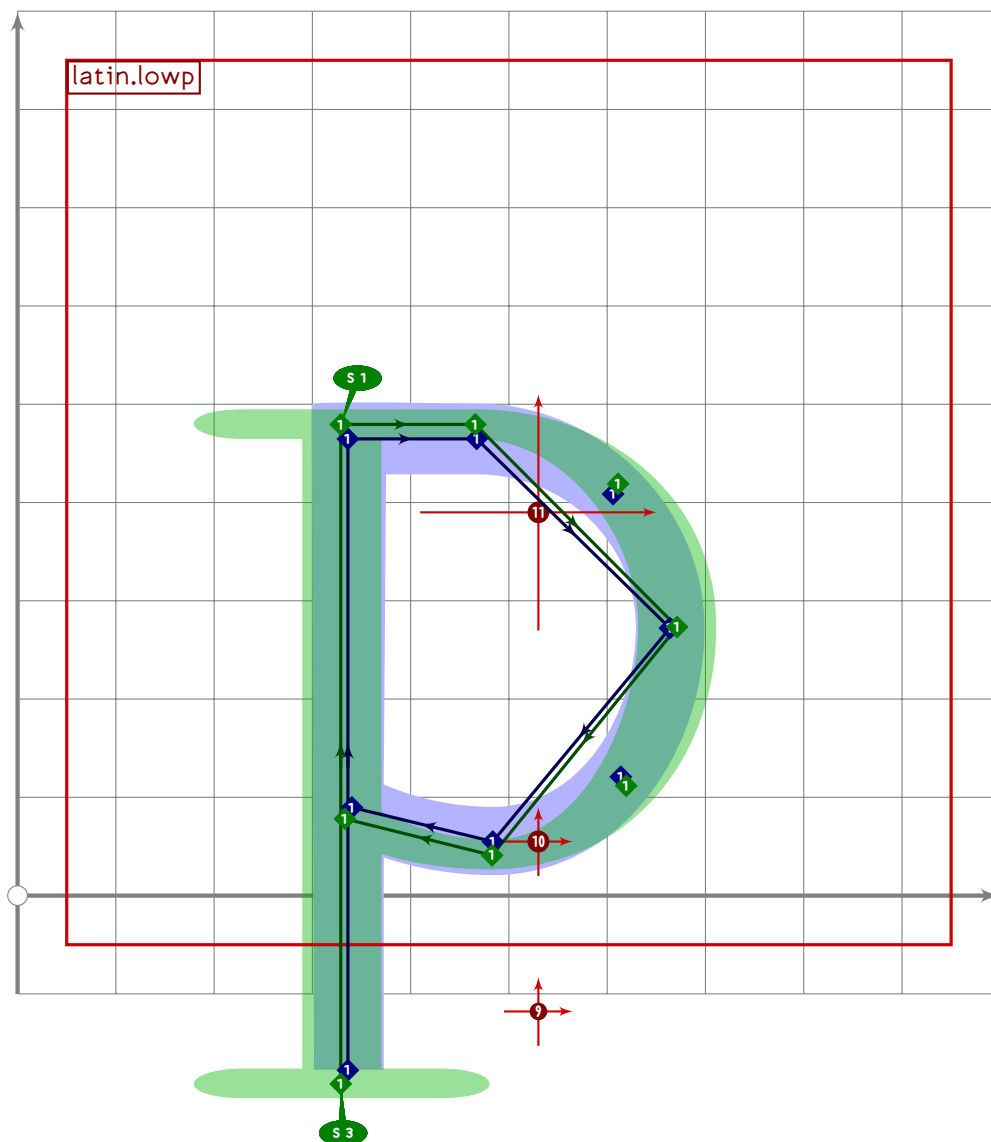
```

1483
1484 vardef latin.lowo =
1485   push_pbox_toexpand("latin.lowo");
1486   push_anchor(anc_centre,identity
1487     scaled ((latin_wide_xheight_r-latin_wide_low_r)/200)
1488     shifted (xpart centre_pt,(latin_wide_xheight_r+latin_wide_low_r)/2));
1489   push_stroke(((1,0)..(0,1)..(-1,0)..(0,-1)..cycle)
1490     scaled ((latin_wide_xheight_r-latin_wide_low_r)/2)
1491     shifted (xpart centre_pt,(latin_wide_xheight_r+latin_wide_low_r)/2),
1492     (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-cycle);
1493   expand_pbox;
1494 enddef;

```

**LATI**





```

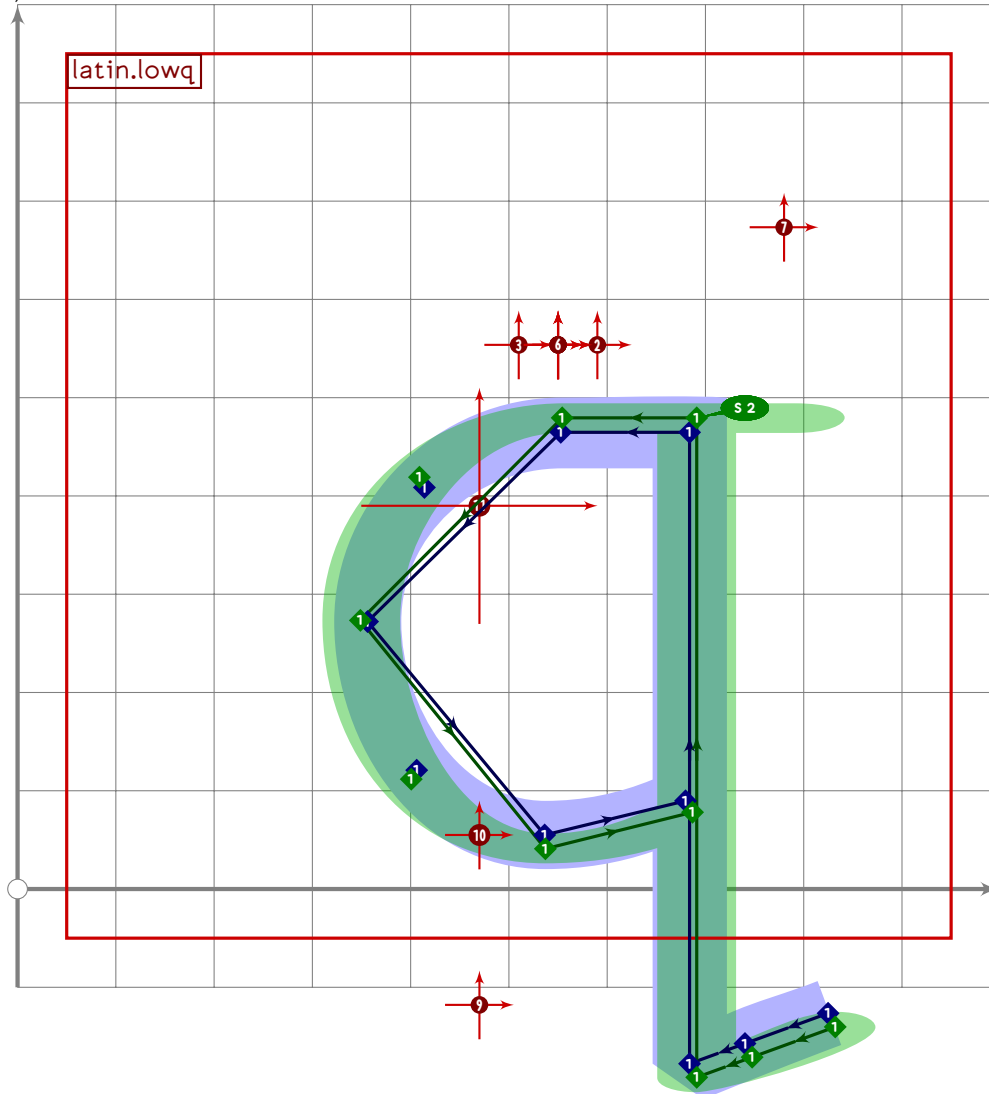
1510
1511 vardef latin.lowp =
1512   push_pbox_toexpand("latin.lowp");
1513   (x1+x4)/2=500;
1514   (x4-x1)=(y2-y1)*0.51;
1515   x2=x1=x6;
1516   x3=0.4[x2,x4];
1517   x5=0.45[x2,x4];
1518
1519   y1=latin_wide_desc_v;
1520   y2=y3=latin_wide_xheight_h;
1521   y4=0.47[y3,y5];
1522   y5=latin_wide_low_h;
1523   y6=0.91[y3,y5];
1524
1525   push_stroke(z1-z2{right}..{right}z3.{down}z4.{left}z5..z6,
1526     (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-
1527     (1.6,1.6)-(1.6,1.6));

```

```

1528 replace_strokep(0)(subpath (0,4.97) of oldp);
1529 set_botip(0,1,1);
1530 if not do_italic_hook: set_boserif(0,0,3); fi;
1531 set_boserif(0,1,1);
1532-1533
1534 tsu_accent.shift_anchors(ypart olda<vmetric(0.52))((30,0));
1535 expand_pbox;
1536 enddef;

```



```

1537
1538 vardef latin.lowq =
1539   push_pbox_toexpand("latin.lowq");
1540   (x1+x4)/2=520;
1541   (x1-x4)=(y2-y1)*0.51;
1542   x2=x1=x6;
1543   x3=0.4[x2,x4];
1544   x5=0.45[x2,x4];
1545
1546   y1=latin_wide_desc_y;

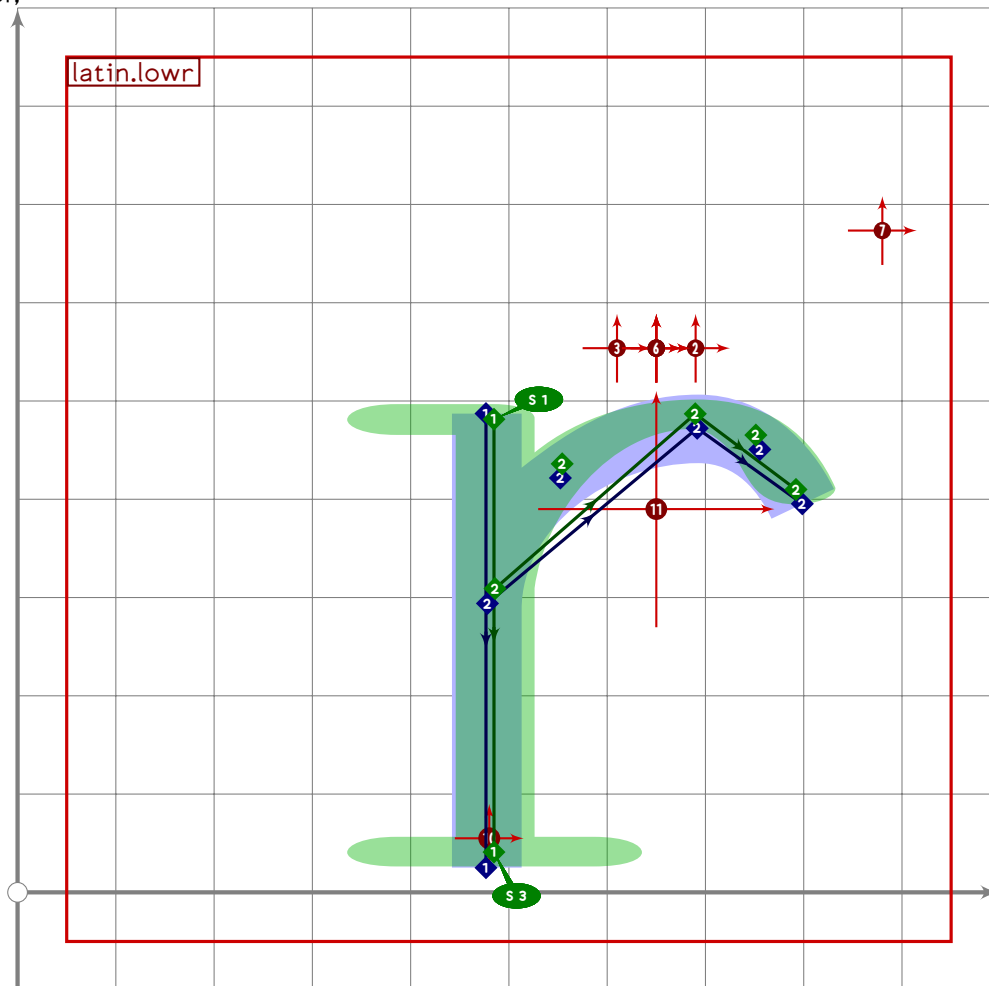
```

U+FF52  
tsuku.uniFF52

```

1547 y2=y3=latin_wide_xheight_h;
1548 y4=0.47[y3,y5];
1549 y5=latin_wide_low_h;
1550 y6=0.91[y3,y5];
1551
1552 z0=z1+150*(dir 20);
1553
1554 push_stroke(z0-(0.6[z0,z1])-z1-z2{left}..
1555   {left}z3..{down}z4..{right}z5..z6,
1556   (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-
1557   (1.6,1.6)-(1.6,1.6)-(1.6,1.6));
1558 replace_strokep(0)(subpath (0,6.97) of oldp);
1559 set_botip(0,2,0);
1560 set_botip(0,3,1);
1561 if not do_italic_hook: set_boserif(0,3,2); fi;
1562
1563 tsu_accent.shift_anchors(ypart olda<vmetric(0.52))((-30,0));
1564 tsu_accent.shift_anchors(ypart olda>vmetric(0.52))((50,0));
1565 expand_pbox;
1566 enddef;

```



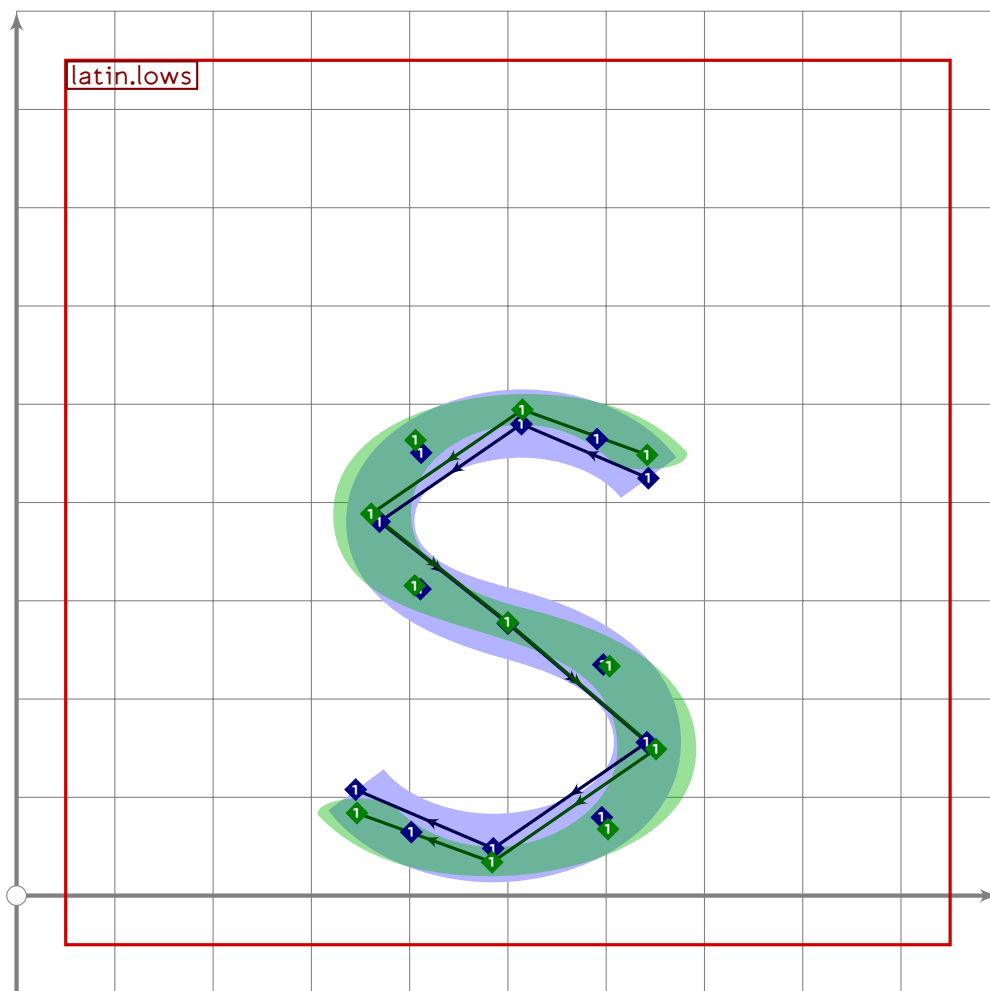
LATI

1567

```

1568 vardef latin.lowr =
1569   push_pbox_toexpand("latin.lowr");
1570   (x1+x5)/2=650;
1571   (x5-x1)=(y1-y2)*0.75;
1572   x2=x1=x3;
1573   x4=0.62[x3,x5];
1574
1575   y1=latin_wide_xheight_v;
1576   y2=latin_wide_low_v;
1577   y3=0.58[y2,y4];
1578   y4=0.5[latin_wide_xheight_h,latin_wide_xheight_r];
1579   y5=0.60[y2,y4];
1580
1581   push_stroke(z1-z2,(1.6,1.6)-(1.6,1.6));
1582   set_boserif(0,0;if do_italic_hook: 11 else: 1 fi);
1583   if not do_italic_hook: set_boserif(0,1,3); fi;
1584
1585   push_stroke(z3..z4{right}..{dir 273}z5,
1586     (1.6,1.6)-(1.6,1.6)-(1.6,1.6));
1587   replace_strokep(0)(subpath (0.03,1.6) of oldp);
1588
1589   tsu_accent.shift_anchors(y part olda>vmetric(0.05))((0.5[x3,x5]-500,0));
1590   tsu_accent.shift_anchors(ai=anc_lower_connect)((-20,0));
1591   expand_pbox;
1592 enddef;

```



```

1593
1594 vardef latin.lows =
1595   push_pbox_toexpand("latin.lows");
1596   transform tatb;
1597   path mycurve;
1598
1599   mycurve:=(1,0)..(0,1)..(-1,0);
1600
1601   y2=latin_wide_xheight_r;
1602   y0=y3=0.77[y6,y2];
1603   y4=0.53[y6,y2];
1604   y5=y8=0.25[y6,y2];
1605   y6=latin_wide_low_r;
1606
1607   0.48[x1,x7]=0.48[x2,x6]=0.48[x3,x5]=x4=500;
1608   x5-x1=5;
1609   x5-x7=(y2-y6)*0.67;
1610
1611   (point 0 of mycurve) transformed ta=z0;
1612   (point 0.35 of mycurve) transformed ta=z1;
1613   (point 1 of mycurve) transformed ta=z2;

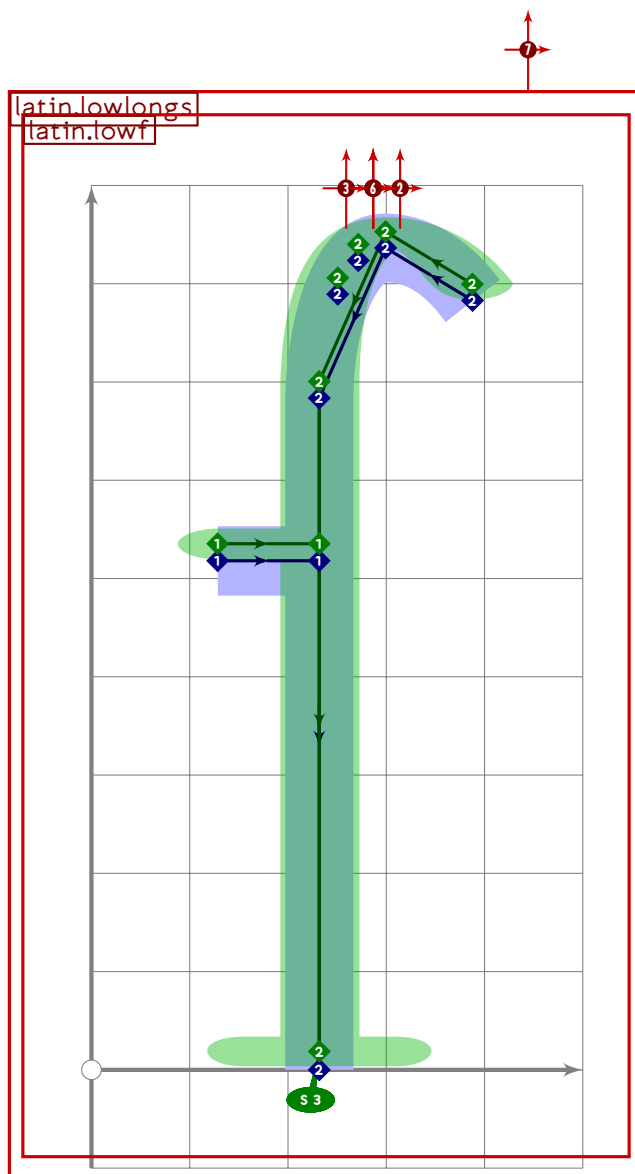
```

```

1614 (point 2 of mycurve) transformed ta=z3;
1615 xypart ta=0;
1616
1617 (point 0 of mycurve) transformed tb=z8;
1618 (point 0.35 of mycurve) transformed tb=z7;
1619 (point 1 of mycurve) transformed tb=z6;
1620 (point 2 of mycurve) transformed tb=z5;
1621
1622 if sharp_corners:
1623   mycurve:=subpath (0.29,2) of mycurve;
1624 else:
1625   mycurve:=subpath (0.38,2) of mycurve;
1626 fi;
1627
1628 push_stroke((mycurve transformed ta)..z4..(reverse mycurve transformed tb),
1629   (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-
1630   (1.6,1.6)-(1.6,1.6));
1631 expand_pbox;
1632 enddef;

```

U+017F  
tsuku.long

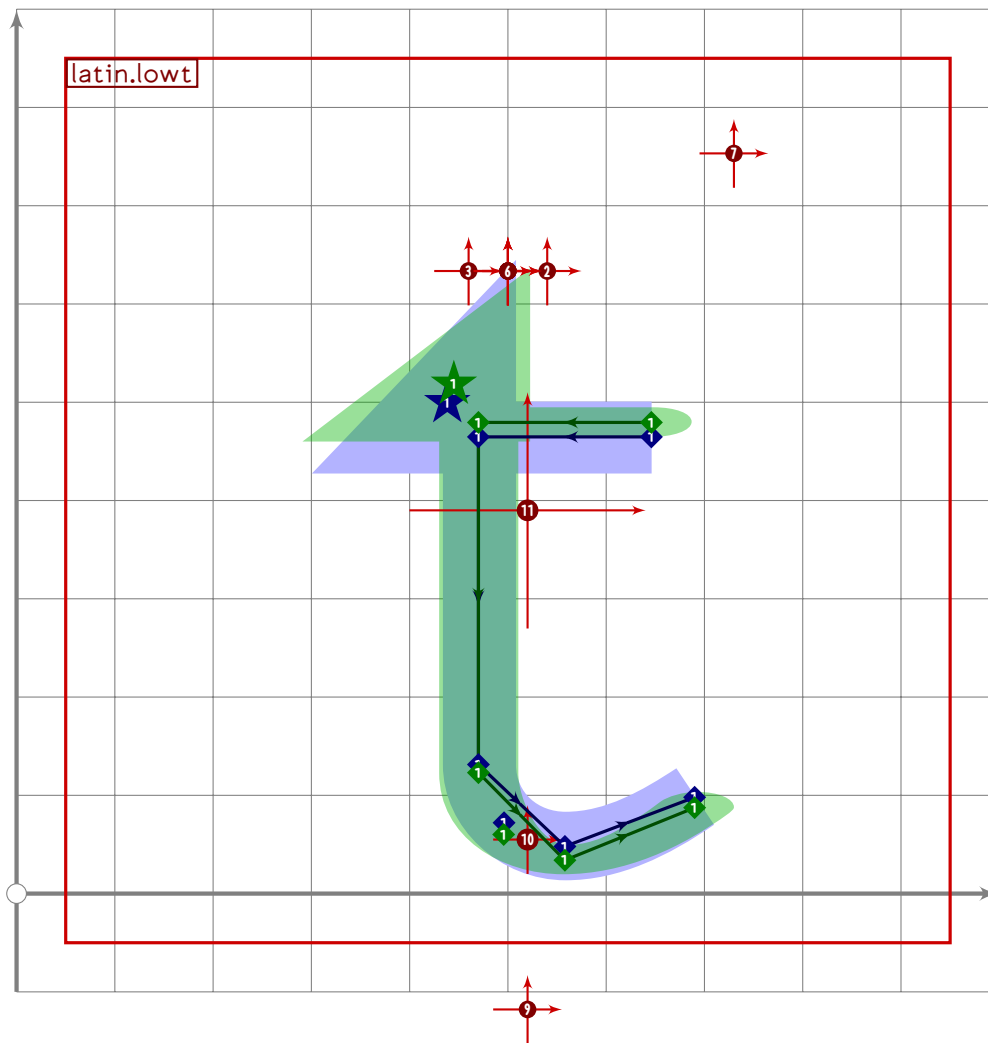


```

1633
1634 vardef latin.longlongs =
1635   push_pbox_toexpand("latin.longlongs");
1636   latin.lowf;
1637   replace_stroke(-1)(subpath (0,0.52) of oldp);
1638   expand_pbox;
1639 enddef;

```

LATI



```

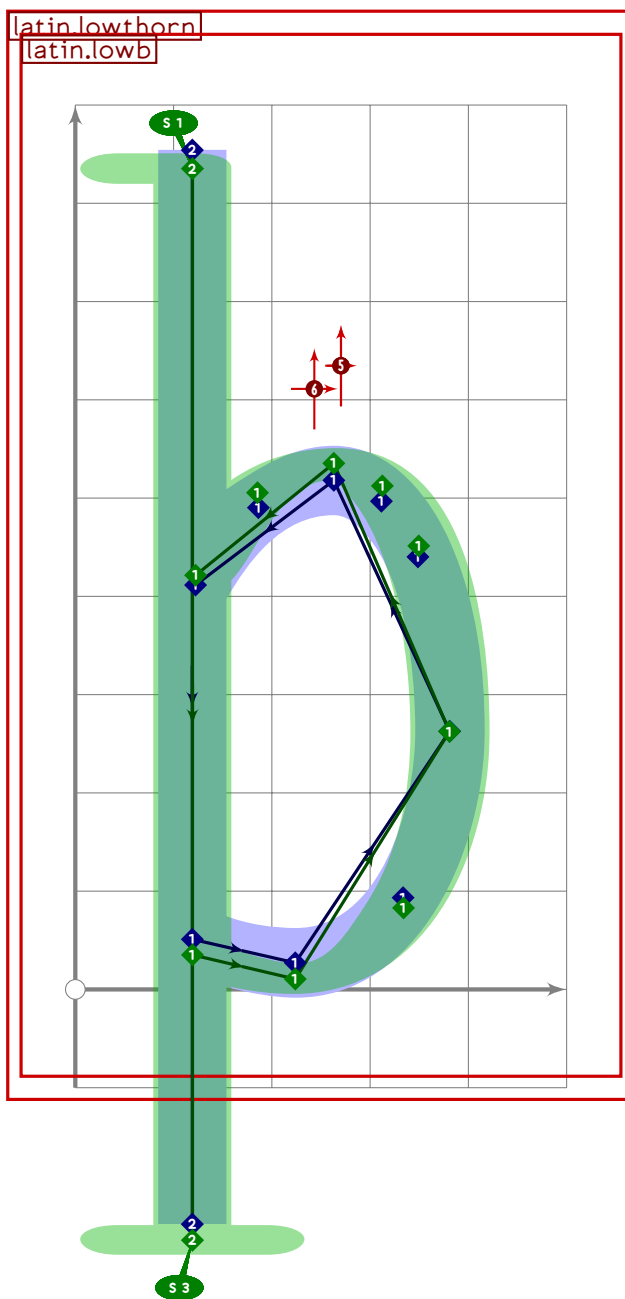
1640
1641 vardef latin.lowt =
1642   push_pbox_toexpand("latin.lowt");
1643   x2=x3=x10=470;
1644   x1=0.2[x5,x2];
1645   x4=0.6[x5,x2];
1646   x5-x2=220;
1647
1648   y1=y2=y6=latin_wide_xheight_h;
1649   y3=0.2[y4,y1];
1650   y4=latin_wide_low_r;
1651   y5=0.12[y4,y1];
1652   y10=vmetric(0.83);
1653
1654   if is_proportional:
1655     z10-z6=whatever*dir 58;
1656   else:
1657     z10-z6=whatever*dir 47;
1658   fi;
1659

```

```

1660 if tsu_pbrush_size<30:
1661     push_stroke(z1-z6-z10-z3{down}..z4{right}..{curl 0.2}z5,
1662         (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
1663     set_botip(0,1,1);
1664     set_botip(0,2,1);
1665 else:
1666     push_stroke(z1-z2-z3{down}..z4{right}..{curl 0.2}z5,
1667         (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
1668     set_botip(0,1,0);
1669 fi;
1670
1671 push_stroke(z6-z1,(0,0)-(0,0));
1672
1673 x7=x8=x2;
1674 x9=x6;
1675
1676 y7=y9=y2;
1677
1678 if is__proportional:
1679     z8-z9=whatever*dir 58;
1680 else:
1681     z8-z9=whatever*dir 47;
1682 fi;
1683
1684 if tsu_pbrush_size>=30:
1685     begingroup
1686         save t; transform t;
1687         t:=tsu_rescale__xform;
1688         push_lcblob(((z7 transformed t)+(mbrush__width,mbrush__height))-
1689             ((z8 transformed t)+(mbrush__width,mbrush__height))-
1690             ((z9 transformed t)+(-mbrush__width,-mbrush__height))-cycle);
1691         replace_lcblob(0)(oldblob transformed inverse t);
1692     endgroup;
1693 fi;
1694
1695 tsu__accent.shift_anchors(ypart olda>vmetric(0.52))
1696     ((0,vmetric(0.12)-vmetric(0)));
1697 tsu__accent.shift_anchors(ypart olda<vmetric(0.52))
1698     ((20,0));
1699 expand_pbox;
1700 endif;

```



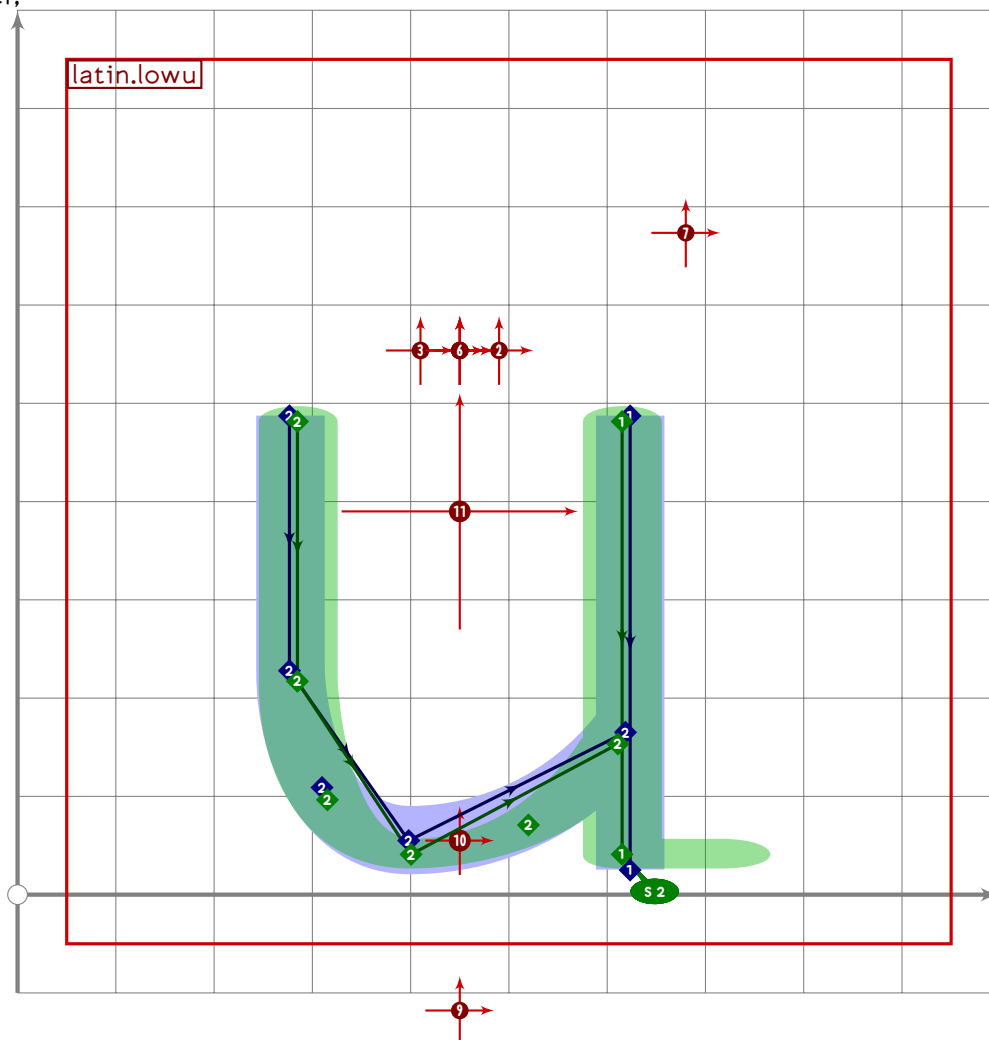
```

1701
1702 vardef latin.lowthorn =
1703   push_pbox_toexpand("latin.lowthorn");
1704   latin.lowb;
1705   set_botip(0,1,whatever);
1706   set_boserif(0,0,whatever);
1707   push_stroke((point 0 of get_strokep(0))--
1708     (xpart point 0 of get_strokep(0),latin_wide_desc_v),
1709     (1.6,1.6)--(1.6,1.6));
1710   set_boserif(0,0,1);
1711   set_boserif(0,1,3);
1712   replace_strokep(-1)(subpath (1,infinity) of oldp);
1713   replace_strokeq(-1)(subpath (1,infinity) of oldq);
1714   expand_pbox;

```

U+FF55  
tsuku.uniFF55

1715 enddef;



```

1716
1717 vardef latin.lowu =
1718   push_pbox_toexpand("latin.lowu");
1719   (x1+x5)/2=450;
1720   (x1-x5)=(y2-y1)*0.75;
1721   x2=x1=x3;
1722   x4=0.65[x3,x5];
1723   x6=x5;
1724
1725   y1=latin_wide_low_v;
1726   y2=y6=latin_wide_xheight_v;
1727   y3=0.73[y2,y4];
1728   y4=latin_wide_low_h;
1729   y5=0.60[y2,y4];
1730
1731   push_stroke(z2-z1,(1.6,1.6)-(1.6,1.6));
1732   set_boserif(0,if do_italic_hook: 11 else: 2 fi);
1733
1734   push_stroke(reverse(z3..z4{left}..z5{dir 93}-z6),

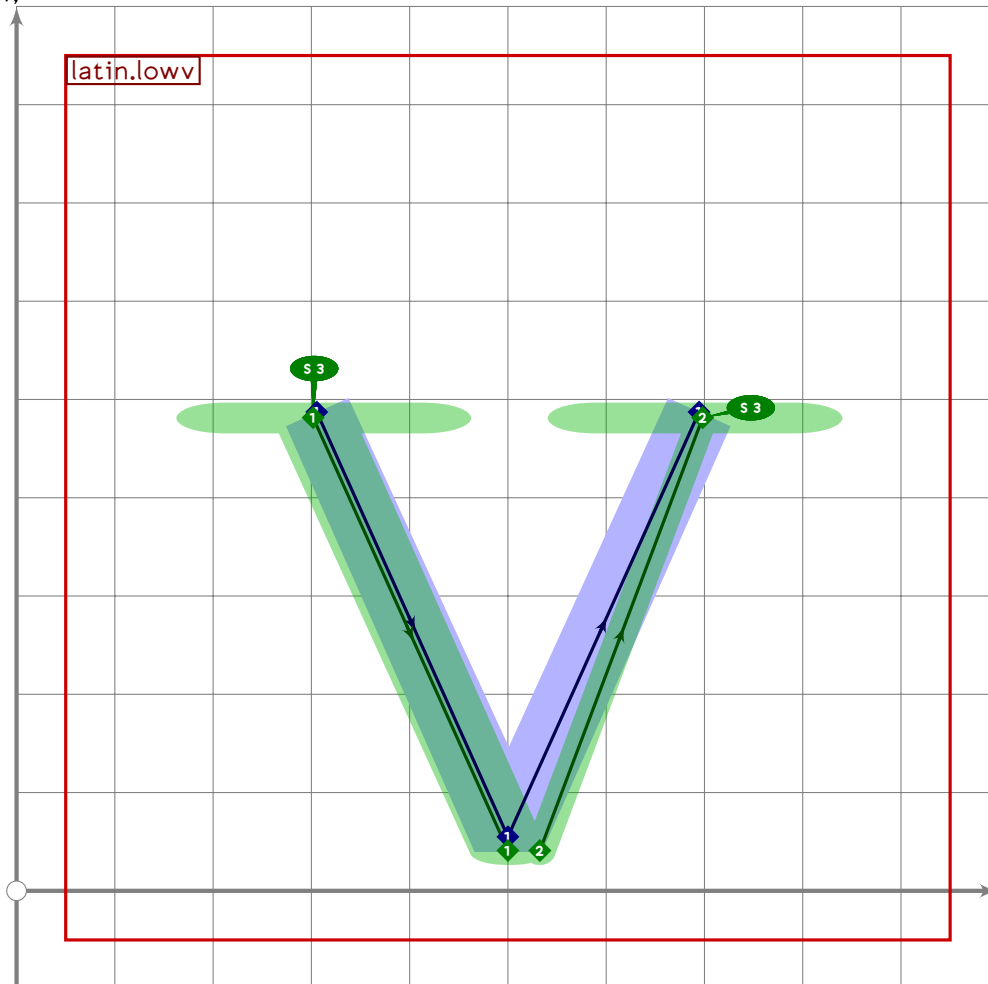
```

LATI

```

1735     (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
1736 replace_stroke(0)(subpath (0,2.97) of oldp);
1737 if do_italic_hook: set_boserif(0,0,11); fi;
1738
1739 tsu_accent.shift_anchors(true)((-50,0));
1740 expand_pbox;
1741 enddef;

```



```

1742
1743 vardef latin.lowv =
1744   push_pbox_toexpand("latin.lowv");
1745   (x1+x3)/2=x2=500;
1746
1747   y1=y3=latin_wide_xheight_v;
1748   y2=latin_wide_low_h;
1749
1750   (x3-x1)=(y1-y2)*0.9;
1751
1752   if do_alteration:
1753     push_stroke(z1-z2,(1.6,1.6)-(1.6,1.6));
1754     set_boserif(0,0,3);
1755

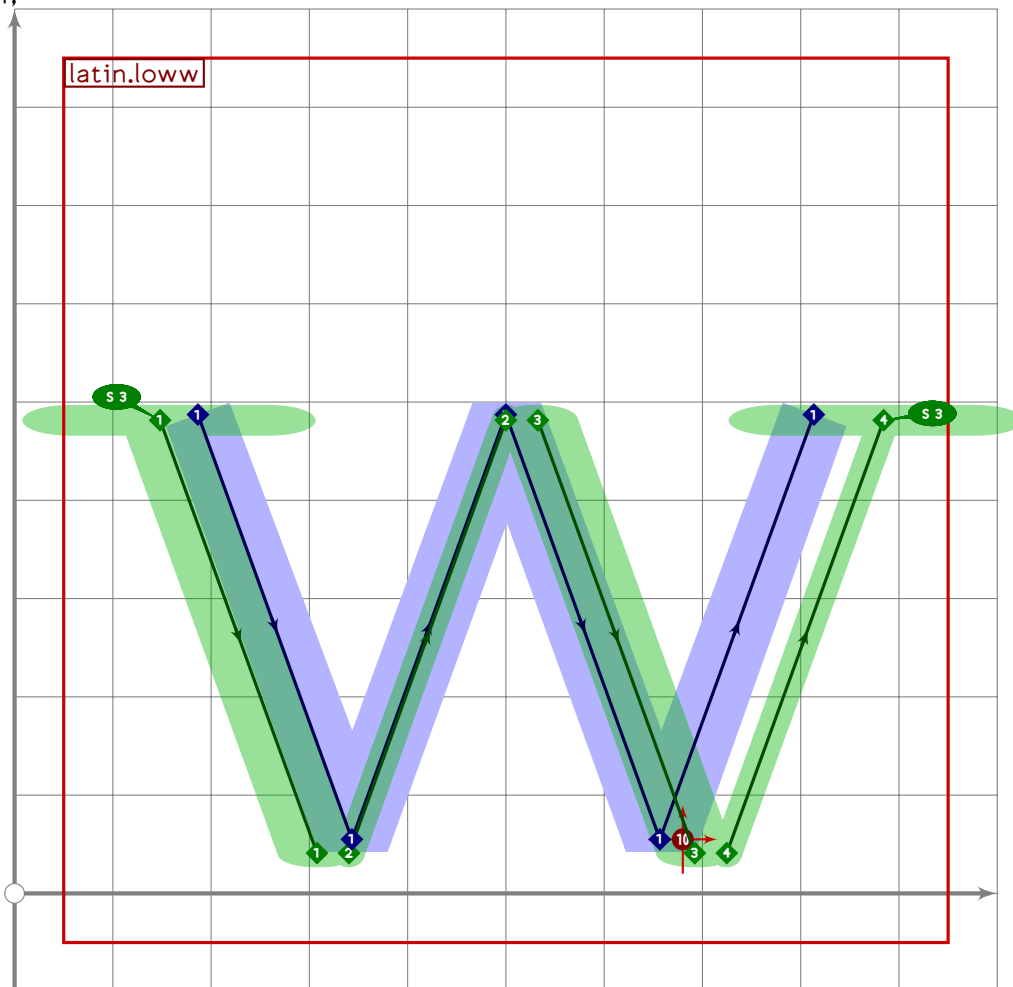
```

U+FF57  
tsuku.uniFF57

```

1756   push_stroke((z2+alternate_adjust*right)-z3,(1.6,1.6)-(1.6,1.6));
1757   set_boserif(0,1,3);
1758   set_boalternate(0);
1759   else:
1760     push_stroke(z1-z2-z3,(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
1761     set_botip(0,1,0);
1762     set_boserif(0,0,1);
1763   fi;
1764   expand_pbox;
1765 enddef;

```



```

1766
1767 vardef latin.loww =
1768   push_pbox_toexpand("latin.loww");
1769   (x1+x5)/2=(x2+x4)/2=x3=500;
1770   (x3-x2)=(x2-x1);
1771
1772   y1=y3=y5=latin_wide_xheight_v;
1773   y2=y4=latin_wide_low_h;
1774
1775   (x5-x1)=(y1-y2)*1.45;
1776

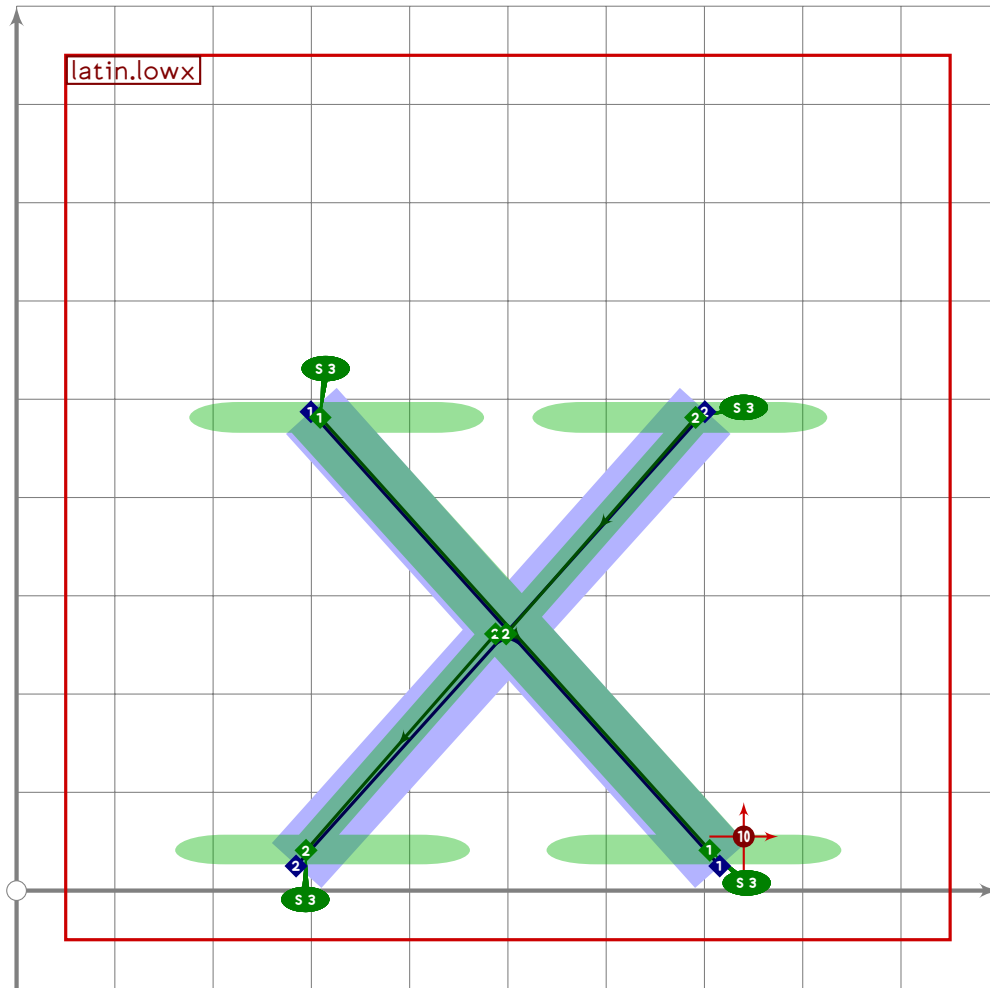
```

**LATI**

```

1777 if do_alternation:
1778     push_stroke((z1-z2) shifted (alternate_adjust*left),
1779         (1.6,1.6)-(1.6,1.6));
1780     set_boserif(0,0,3);
1781
1782     push_stroke(z2-z3,
1783         (1.6,1.6)-(1.6,1.6));
1784     set_boalternate(0);
1785
1786     push_stroke((z3-z4) shifted (alternate_adjust*right),
1787         (1.6,1.6)-(1.6,1.6));
1788
1789     push_stroke((z4-z5) shifted (alternate_adjust*right*2),
1790         (1.6,1.6)-(1.6,1.6));
1791     set_boserif(0,1,3);
1792     set_boalternate(0);
1793 else:
1794     push_stroke(z1-z2-z3-z4-z5,
1795         (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
1796     set_botip(0,1,0);
1797     set_botip(0,2,0);
1798     set_botip(0,3,0);
1799     set_boserif(0,0,1);
1800 fi;
1801
1802 tsu_accent.shift_anchors(ai=anc_lower_connect)((180,0));
1803 expand_pbox;
1804 enddef;

```



```

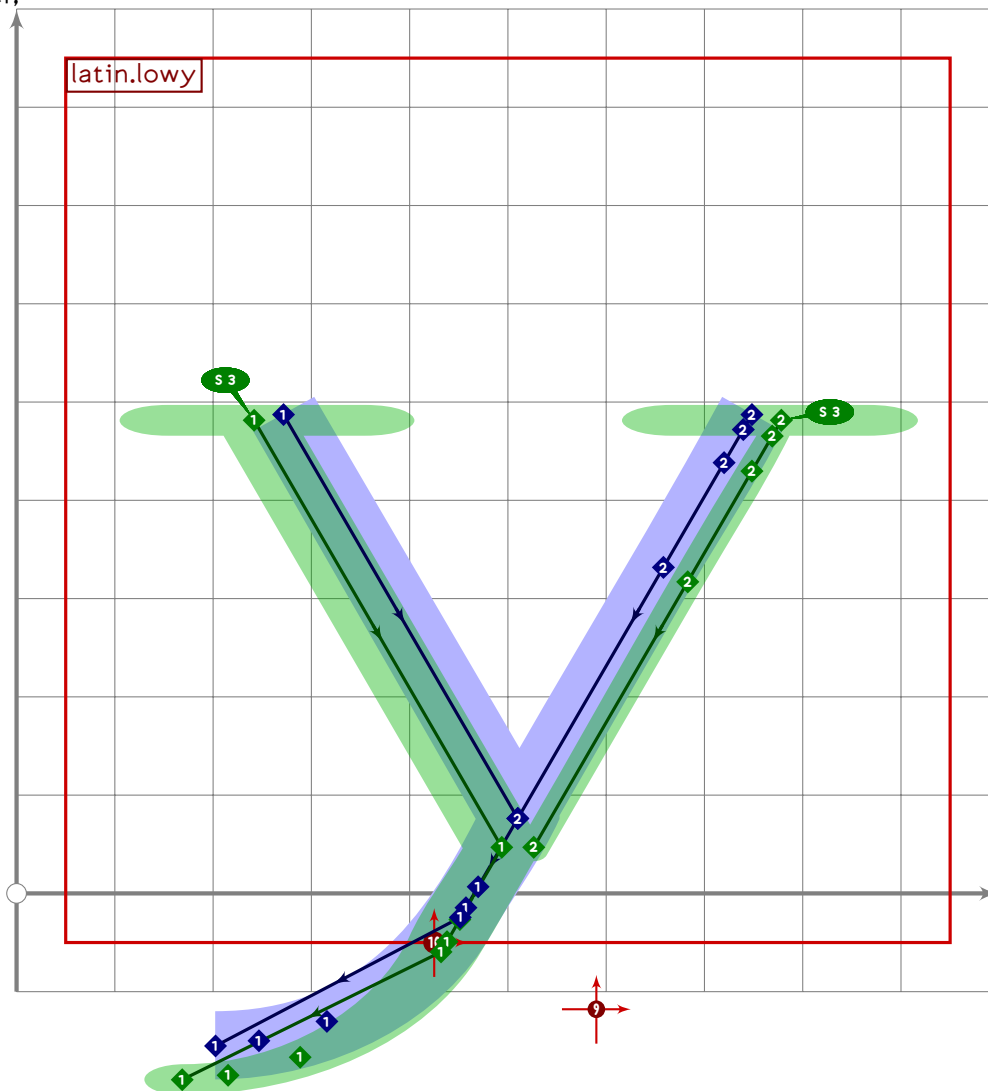
1805
1806 vardef latin.lowx =
1807   push_pbox_toexpand("latin.lowx");
1808   (x1+x3)/2=500;
1809   (x2+x4)/2=500;
1810   (x2+x3-x1-x4)=((y1-y2)*0.9)*2;
1811   (x3-x1)=(x2-x4)*0.93;
1812
1813   y1=y3=latin_wide_xheight_v;
1814   y2=y4=latin_wide_low_v;
1815
1816   push_stroke(z1-z2,(1.6,1.6)-(1.6,1.6));
1817   set_boserif(0,0,if do_italic_hook: 11 else: 3 fi);
1818   set_boserif(0,1,if do_italic_hook: 11 else: 3 fi);
1819
1820   if do_alteration:
1821     push_stroke(z3-(0.5[z3,z4]+alternate_adjust*right/6)
1822       -(0.5[z3,z4]+alternate_adjust*left/6)-z4,
1823       (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
1824     set_boserif(0,0,if do_italic_hook: 2 else: 3 fi);
1825     set_boserif(0,3,if do_italic_hook: 1 else: 3 fi);

```

```

1826 else:
1827   push_stroke(z3-z4,(1.6,1.6)-(1.6,1.6));
1828   set_boserif(0,0,if do_italic_hook: 2 else: 3 fi);
1829   set_boserif(0,1,if do_italic_hook: 1 else: 3 fi);
1830 fi;
1831 set_boalternate(0);
1832
1833 tsu_accent.shift_anchors(ai=anc_lower_connect)((240,0));
1834 expand_pbox;
1835 enddef;

```



```

1836
1837 vardef latin.lowy =
1838   push_pbox_toexpand("latin.lowy");
1839   (x1+x3)/2=(x2+x4)/2=510;
1840   (x2+x3-x1-x4)=((y1-y2)*0.58)*2;
1841   (x3-x1)=(x2-x4)*0.93;
1842   x5=x4-0.1*(x2-x4);
1843

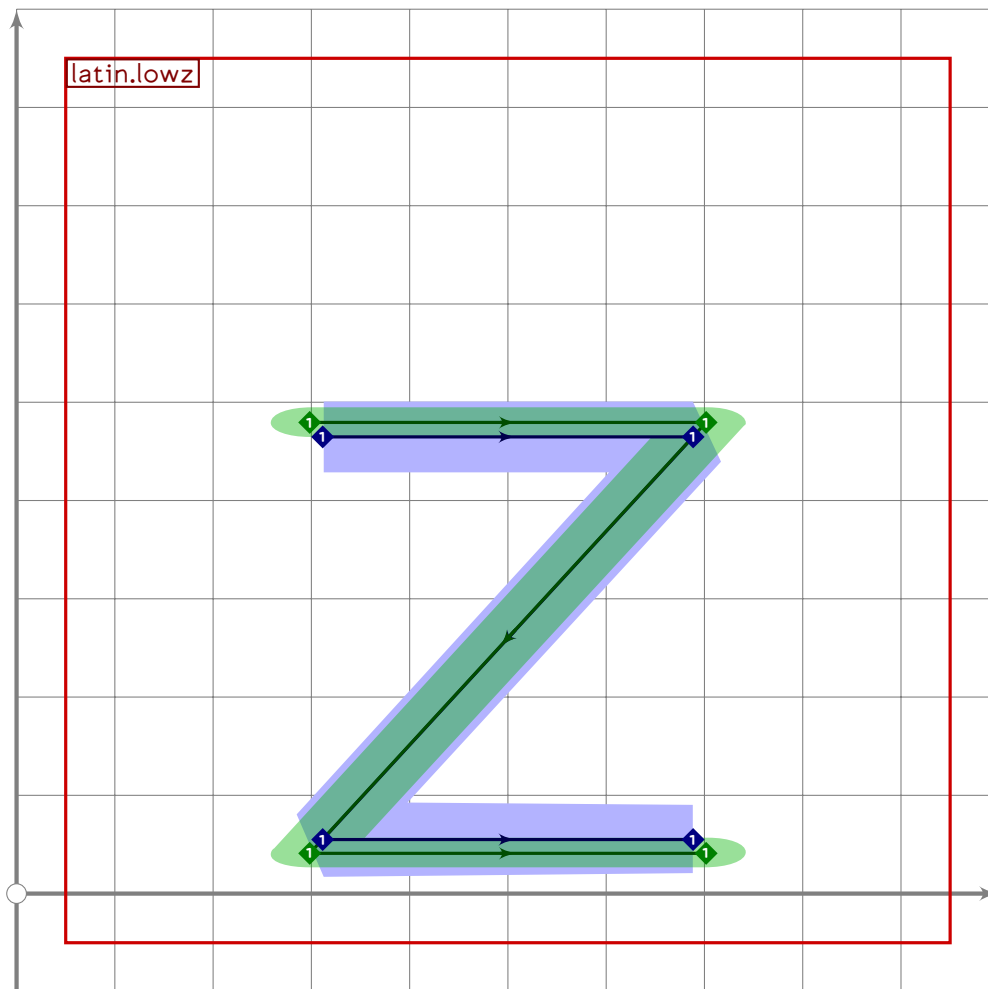
```

LATI

```

1844 y1=y3=latin_wide_xheight_v;
1845 y2=y4;
1846 y5=0.5[y4,latin_wide_low_h]=latin_wide_desc_h;
1847
1848 push_stroke(z1-z2,(1.6,1.6)-(1.6,1.6));
1849
1850 push_stroke(z3..tension 10..(0.6[z3,z4])..tension 0.8 and 3..{left}z5,
1851   (1.6,1.6)-(1.6,1.6)-(1.6,1.6));
1852
1853 numeric xchgtime;
1854 xchgtime:=ypart (get_strokep(-1) intersectiontimes get_strokep(0));
1855
1856 replace_strokep(-1)(z1-subpath (xchgtime,infinity) of get_strokep(0));
1857 replace_strokeq(-1)
1858   ((1.6,1.6)-subpath (xchgtime,infinity) of get_strokeq(0));
1859
1860 replace_strokep(0)(subpath (0,xchgtime) of oldp);
1861 replace_strokeq(0)(subpath (0,xchgtime) of oldq);
1862
1863 set_boserif(-1,0,if do_italic_hook: 11 else: 3 fi);
1864 set_boserif(0,0,if do_italic_hook: 2 else: 3 fi);
1865 set_botip(-1,1,1);
1866 set_boalternate(0);
1867
1868 if do_alteration:
1869   replace_strokep(-1)(oldp shifted (alternate_adjust*left/2));
1870   replace_strokep(0)(oldp shifted (alternate_adjust*right/2));
1871 fi;
1872
1873 tsu_accent.shift_anchors(ai=anc_lower)((90,0));
1874 tsu_accent.shift_anchors(ai=anc_lower_connect)((-75,-105));
1875 expand_pbox;
1876 enddef;

```



```

1877
1878 vardef latin.lowz =
1879   push_pbox_toexpand("latin.lowz");
1880   y1=y2=latin_wide_xheight_h;
1881   y3=y4=latin_wide_low_h;
1882
1883   x1=x3;
1884   x2=x4;
1885   (x1+x2)/2=500;
1886   (x2-x1)=(y1-y3)*0.92;
1887
1888   push_stroke(z1-z2-z3-z4,(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
1889   set_botip(0,1,0);
1890   set_botip(0,2,0);
1891   expand_pbox;
1892 enddef;

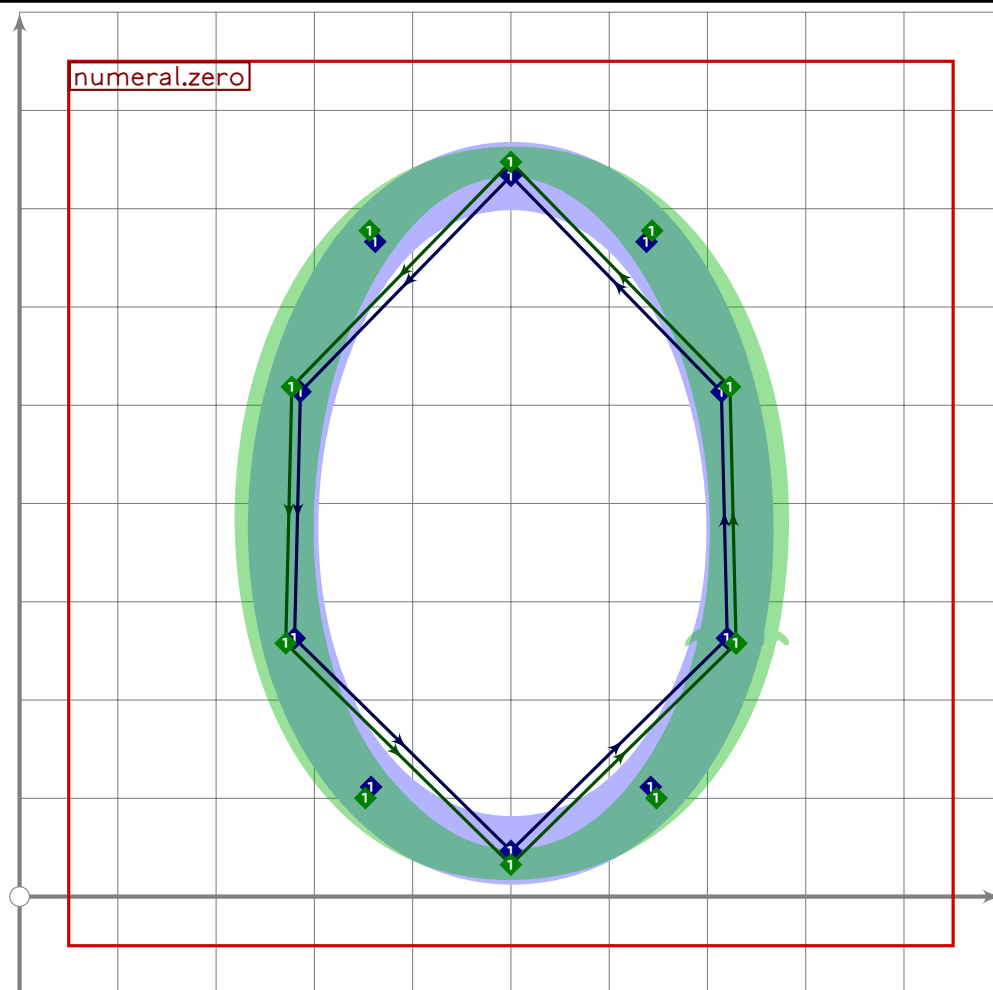
```

# numerals.mp

```

1 %
2 % Hindu/Arabic numerals for Tsukurimashou
3 % Copyright (C) 2011 Matthew Skala
4 %
5-29 [Standard copyright notice]
30
31 inclusion_lock(numeral);
32
33

```

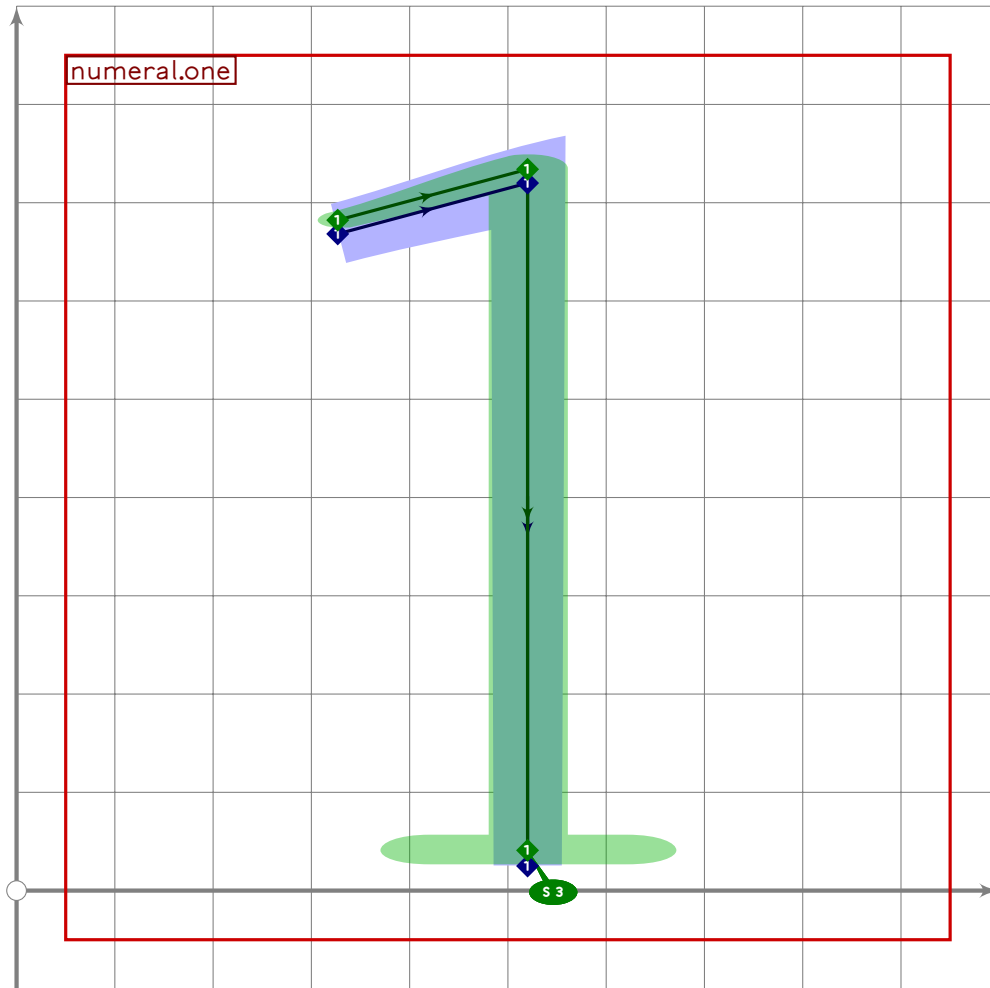


```

34
35 vardef numeral.zero =
36   push_pbox_toexpand("numeral.zero");
37   push_stroke(((0.74*dir 330)..(0.72*dir 30)..(up)..
38     (0.72*dir 150)..(0.74*dir 210)..(down)..cycle)
39     scaled ((latin_wide_high_r-latin_wide_low_r)/2)
40     shifted centre_pt,
41     (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-cycle);
42   expand_pbox;
43 enddef;

```

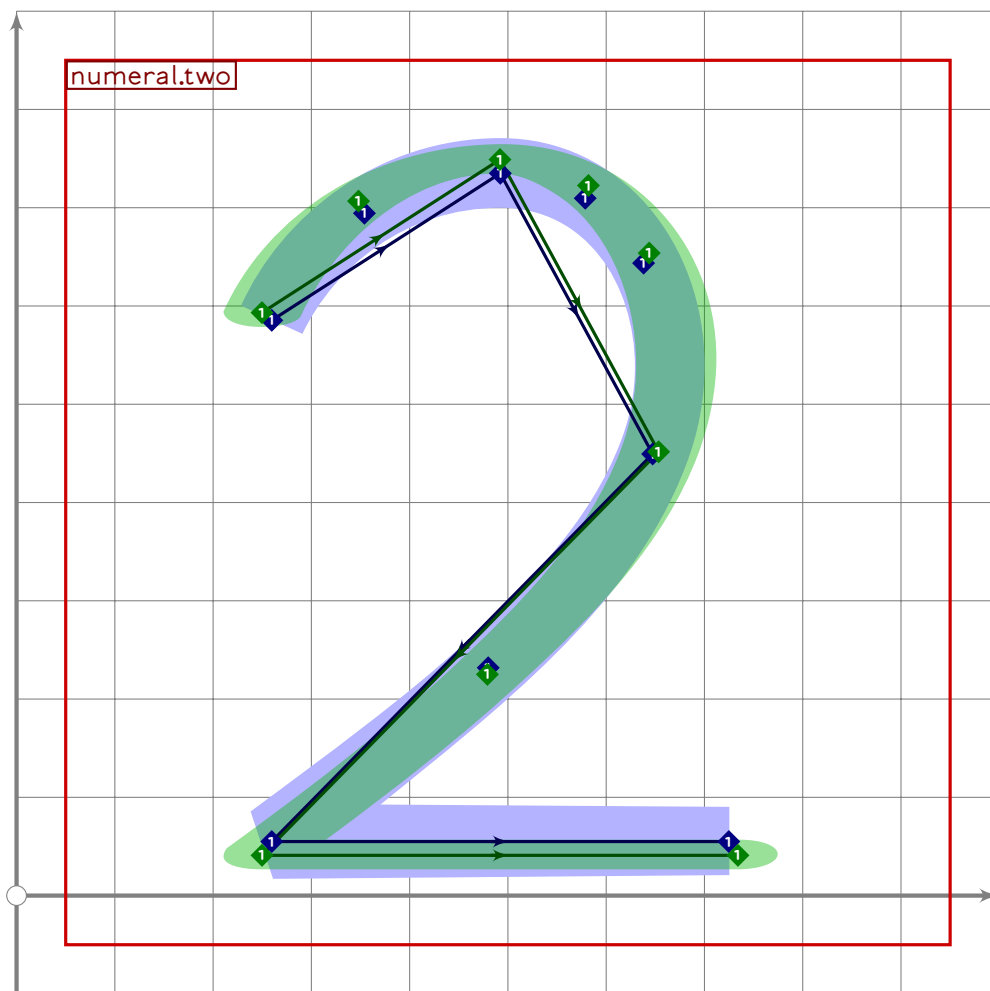
NUME



```

44
45 vardef numeral.one =
46   push_pbox_toexpand("numeral.one");
47   x3=x2=520;
48
49   y2=latin_wide_high_h;
50   y3=latin_wide_low_v;
51
52   z1=z2+200*dir 195;
53
54   push_stroke(z1-z2-z3,(1,1,1)-(1,6,1,6)-(1,6,1,6));
55   set_botip(0,1,1);
56   set_boserif(0,2,3);
57   expand_pbox;
58 enddef;

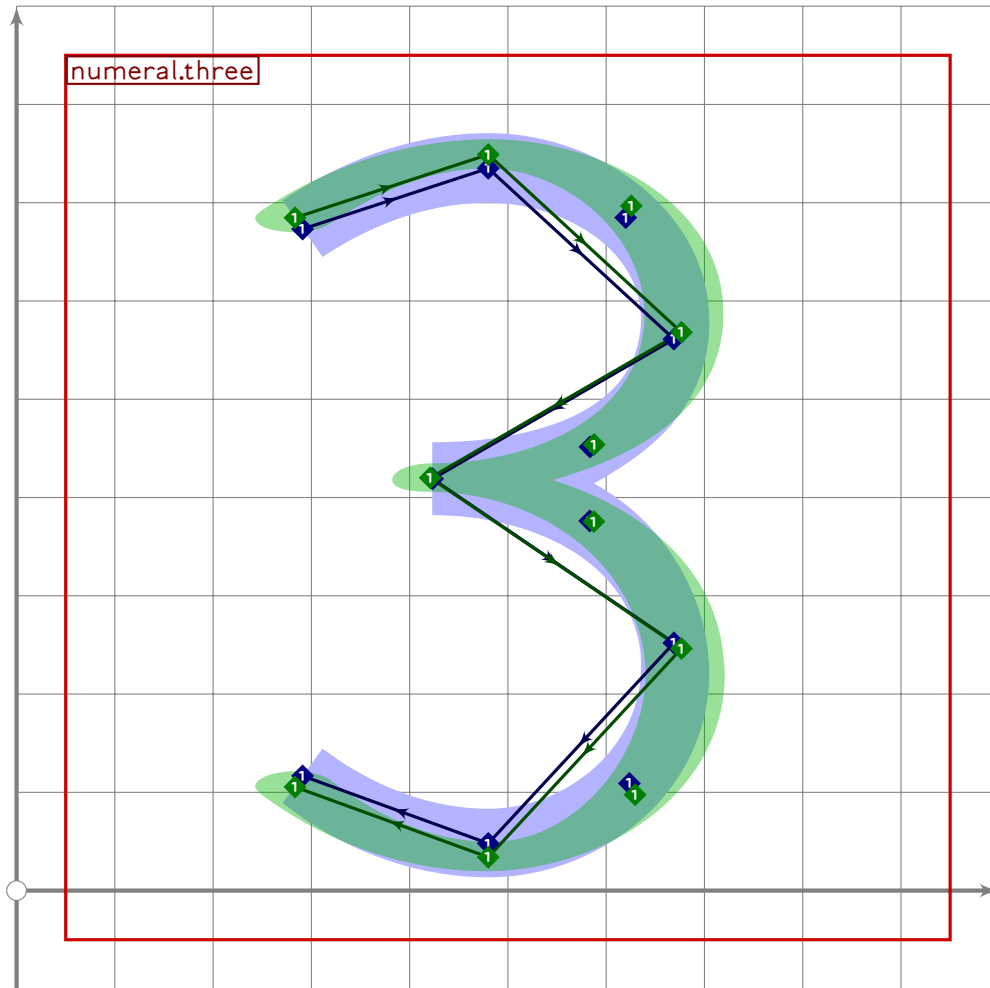
```



```

59
60 vardef numeral.two =
61   push_pbox__toexpand("numeral.two");
62   x1=x4;
63   0.62[x1,x3]=500;
64   x2=0.6[x1,x3];
65   x5=1.2[x1,x3];
66   x3-x1=0.57*(y2-y4);
67
68   y1=0.78[y4,y2];
69   y2=latin__wide__high_r;
70   y3=0.58[y4,y2];
71   y4=y5=latin__wide__low_h;
72
73   push_stroke(z1..z2{right}..z3..tension 1.2..{curl 0}z4-z5,
74     (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
75   set_botip(0,3,0);
76   expand_pbox;
77 enddef;

```



```

78
79 vardef numeral.three =
80   push_pbox_toexpand("numeral.three");
81   x1=x7;
82   x2=x6=0.5[x1,x3];
83   x3=x5;
84   x4=0.35[x1,x3];
85   (x1+x3)/2=480;
86   (x3-x1)=0.55*(y2-y6);
87
88   y1=0.91[y6,y2];
89   y2=latin_wide_high_r;
90   y3=0.45[y4,y2];
91   y4=0.54[y6,y2];
92   y5=0.45[y4,y6];
93   y6=latin_wide_low_r;
94   y7=0.1[y6,y2];
95
96   push_stroke(z1{curl 0.7}..z2{right}..z3.{left}z4{right}..
97     z5..z6{left}..{curl 0.7}z7,
98     (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-

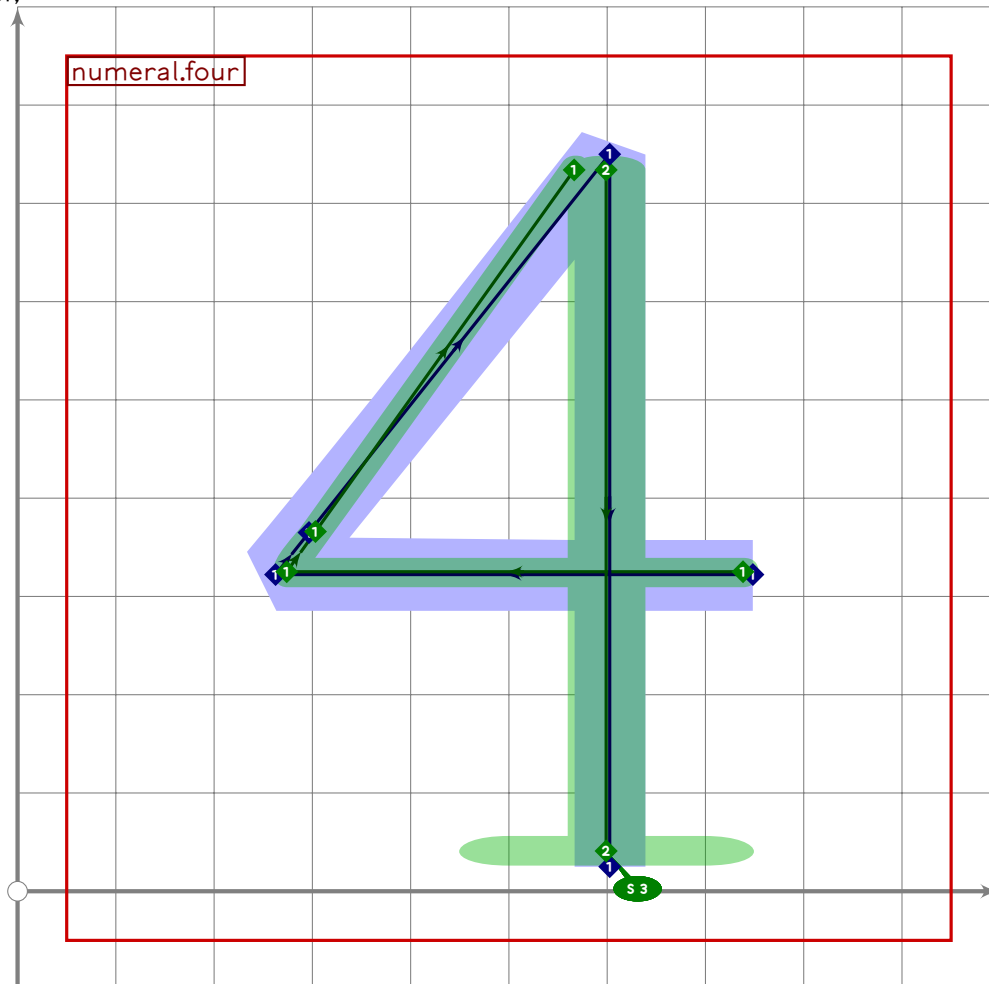
```

U+FF14  
tsuku.uniFF14

```

99      (1.6,1.6)-(1.6,1.6));
100    set_botip(0,3,0);
101    expand_pbox;
102  enddef;

```



```

103
104  vardef numeral.four =
105    push_pbox_toexpand("numeral.four");
106    x3=x4=0.7[x2,x1];
107    0.53[x2,x1]=520;
108    (x1-x2)=0.67(y3-y4);
109
110    y1=y2=0.41[y4,y3];
111    y3=latin_wide_high_v;
112    y4=latin_wide_low_v;
113
114    if do_alteration:
115      push_stroke(z1-z2-(0.1[z2,(z3+alternate_adjust*left)])-
116        (z3+alternate_adjust*left),
117        (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
118      set_botip(0,1,0);
119      set_botip(0,2,0);

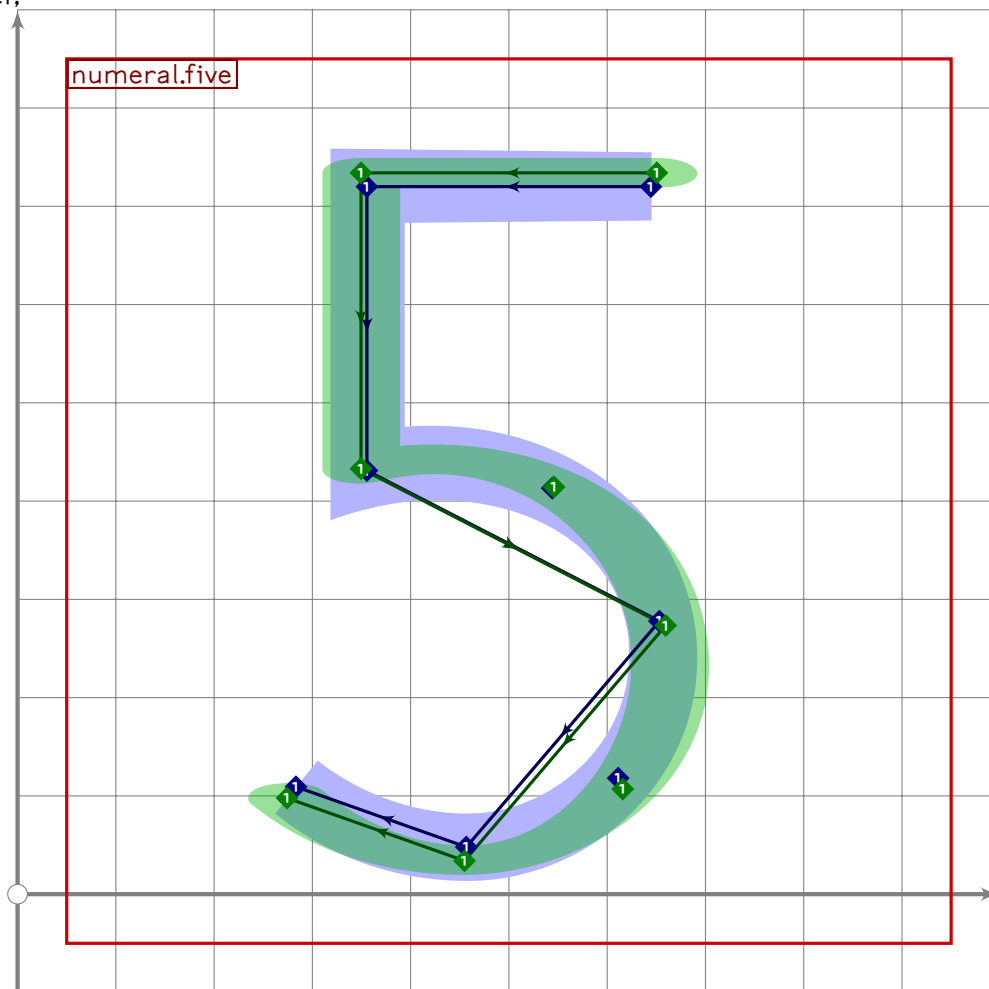
```

NUME

```

120   set_boalternate(0);
121
122   push_stroke(z3-z4,(1.6,1.6)-(1.6,1.6));
123   set_botip(0,0,0);
124   set_boserif(0,1,3);
125   else:
126     push_stroke(z1-z2-(0.1[z2,z3])-z3-z4,
127       (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
128     set_botip(0,1,0);
129     set_botip(0,3,0);
130     set_boserif(0,4,3);
131   fi;
132   expand_pbox;
133 enddef;

```



```

134
135 vardef numeral.five =
136   push_pbox_toexpand("numeral.five");
137   (x1+x2)/2=500;
138   (x1-x2)=(y2-y3);
139   x2=x3;
140   x4=1.03[x2,x1];

```

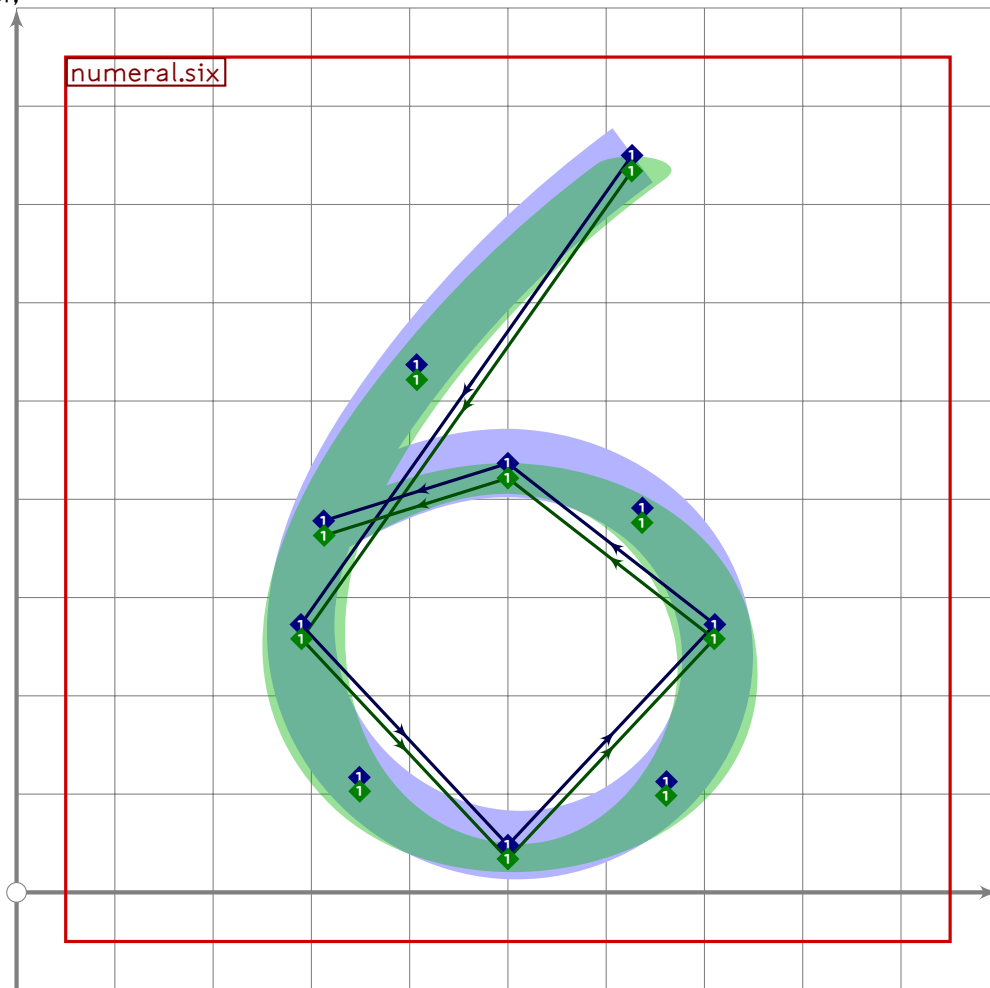
NUME

U+FF16  
tsuku.uniFF16

```

141 x5=0.35[x2,x1];
142 x6=(-0.25)[x2,x1];
143
144 y1=y2=latin_wide_high_h;
145 y3=0.57[y5,y1];
146 y4=0.6[y5,y3];
147 y5=latin_wide_low_r;
148 y6=0.16[y5,y3];
149
150 push_stroke(z1-z2-z3{curl 0.5}..z4..z5{left}..z6,
151   (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
152 set_botip(0,1,1);
153 set_botip(0,2,1);
154 expand_pbox;
155 enddef;

```



NUME

```

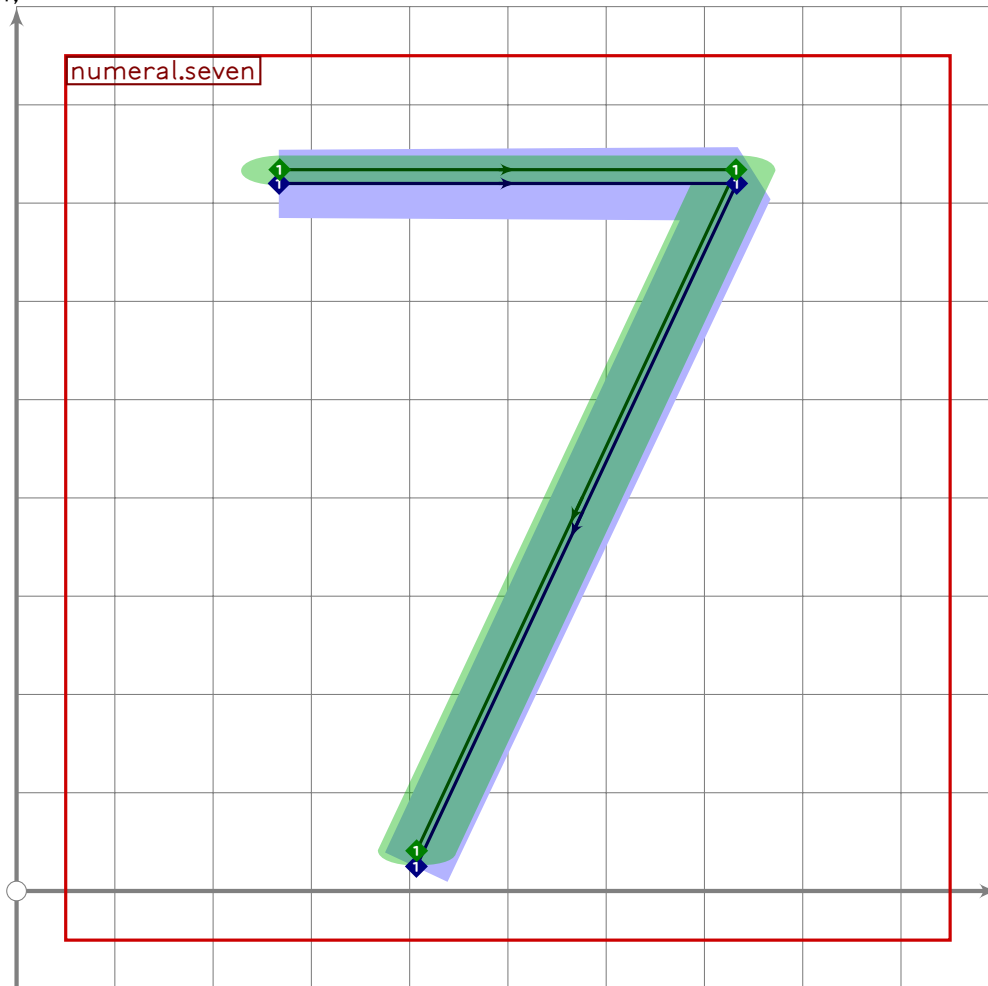
156
157 vardef numeral.six =
158   push_pbox_toexpand("numeral.six");
159   x1=0.8[x2,x4];
160   (x2+x4)/2=x3=500;
161   (x4-x2)=0.6(y1-y3);

```

```

162 x5=x3;
163
164 y1=latin_wide_high_y;
165 y2=y4=0.32[y3,y1];
166 y3=latin_wide_low_r;
167 y5-y4=0.73*(y4-y3);
168
169 push_stroke(z1{curl 0.2}..tension 1.2..z2..z3..z4{dir 100},
170   (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
171 replace_strokep(0)(z1{curl 0.2}..tension 1.2..z2..z3..z4{dir 100}..
172   z5.{curl 0.2}(point 0.8 of oldp));
173 expand_pbox;
174 endif;

```



```

175
176 vardef numeral.seven =
177   push_pbox_toexpand("numeral.seven");
178   (x1+x2)/2=500;
179   x3=0.3[x1,x2];
180   (x2-x1)=0.67*(y1-y3);
181
182   y1=y2=latin_wide_high_h;

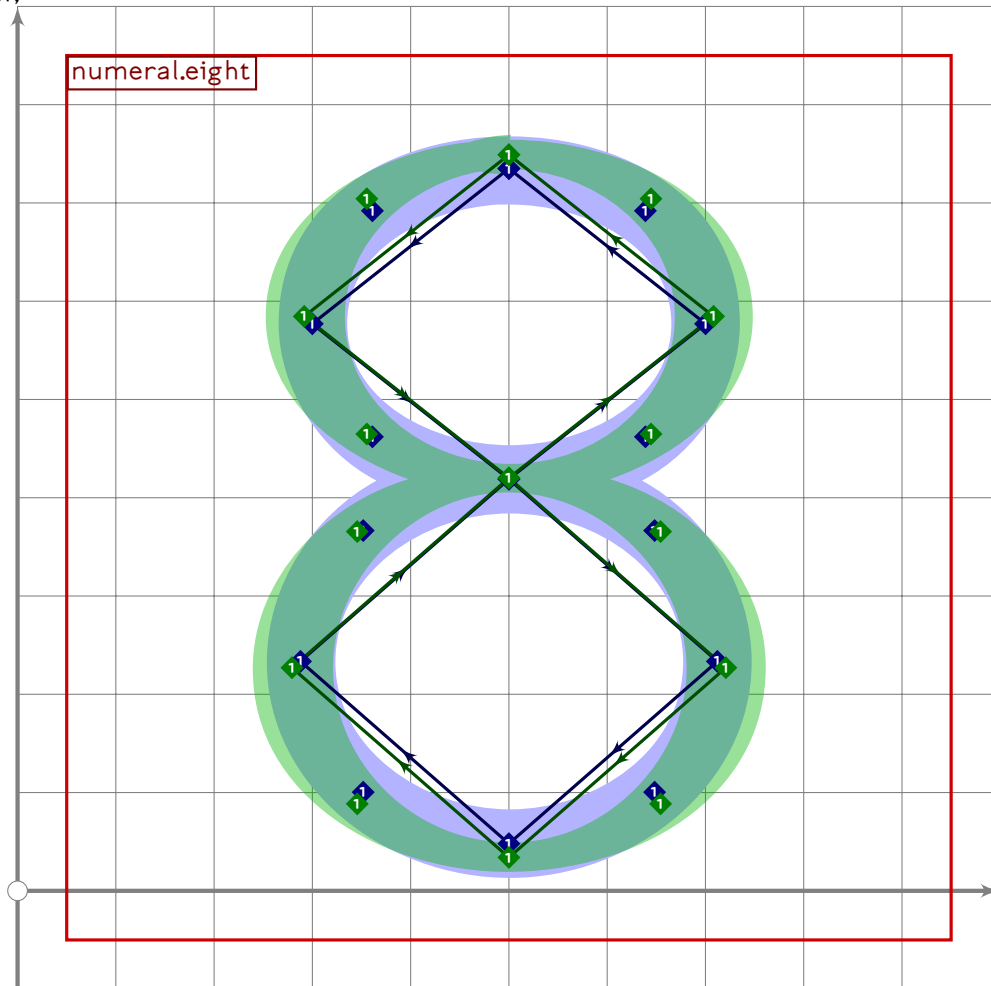
```

U+FF18  
tsuku.uniFF18

```

183 y3=latin_wide_low_v;
184
185 push_stroke(z1-z2-z3,
186   (1.6,1.6)-(1.6,1.6)-(1.6,1.6));
187 set_botip(0,1,0);
188 expand_pbox;
189 enddef;

```



```

190
191 vardef numeral.eight =
192   push_pbox_toexpand("numeral.eight");
193   x1=x3=x5=x7=(x2+x8)/2=(x4+x6)/2=500;
194   (x4-x6)=1.06*(x8-x2);
195   (x4+x8-x6-x2)/2=0.6*(y1-y5);
196
197   y1=latin_wide_high_r;
198   y2=y8=0.5[y3,y1];
199   y3=y7=0.54[y5,y1];
200   y4=y6=0.5[y5,y3];
201   y5=latin_wide_low_r;
202
203   push_stroke(z1..z2..z3{right}..z4..z5..z6..z7{right}..z8..cycle,

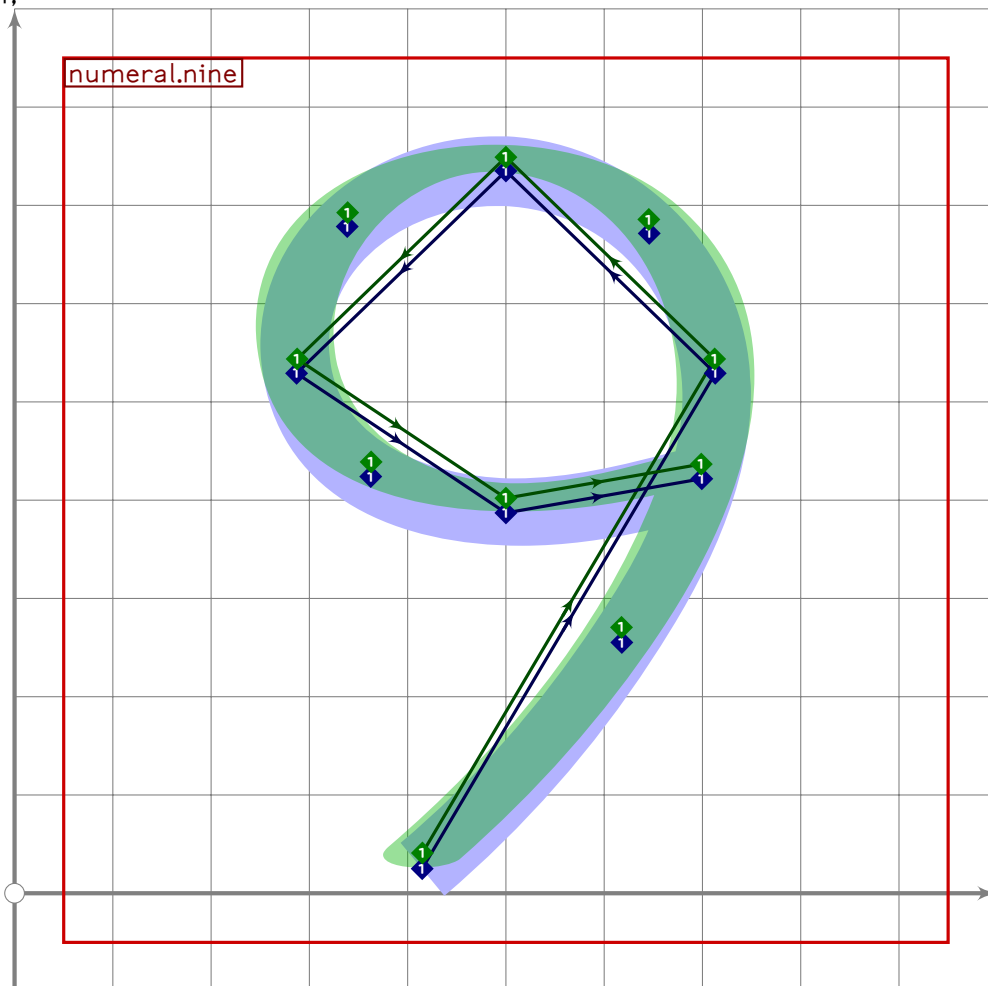
```

NUME

```

204 (1.6,1.6)–(1.6,1.6)–(1.6,1.6)–(1.6,1.6)–(1.6,1.6)–(1.6,1.6)–
205 (1.6,1.6)–(1.6,1.6)–cycle);
206 expand_pbox;
207 enddef;

```



```

208
209 vardef numeral.nine =
210   push_pbox_toexpand("numeral.nine");
211   x1=0.3[x4,x2];
212   (x2+x4)/2=x3=500;
213   (x2-x4)=0.6(y3-y1);
214   x5=x3;
215
216   y1=latin_wide_low_v;
217   y2=y4=0.29[y3,y1];
218   y3=latin_wide_high_r;
219   y5-y4=0.69*(y4-y3);
220
221   push_stroke(z1{curl 0.2}.tension 1.2..z2..z3..z4{dir 280},
222     (1.6,1.6)–(1.6,1.6)–(1.6,1.6)–(1.6,1.6)–(1.6,1.6));
223   replace_strokep(0)(z1{curl 0.2}.tension 1.2..z2..z3..z4{dir 280}..
224     z5..{curl 0.2}(point 0.8 of oldp));

```

```
225   expand_pbox;  
226 endif;
```

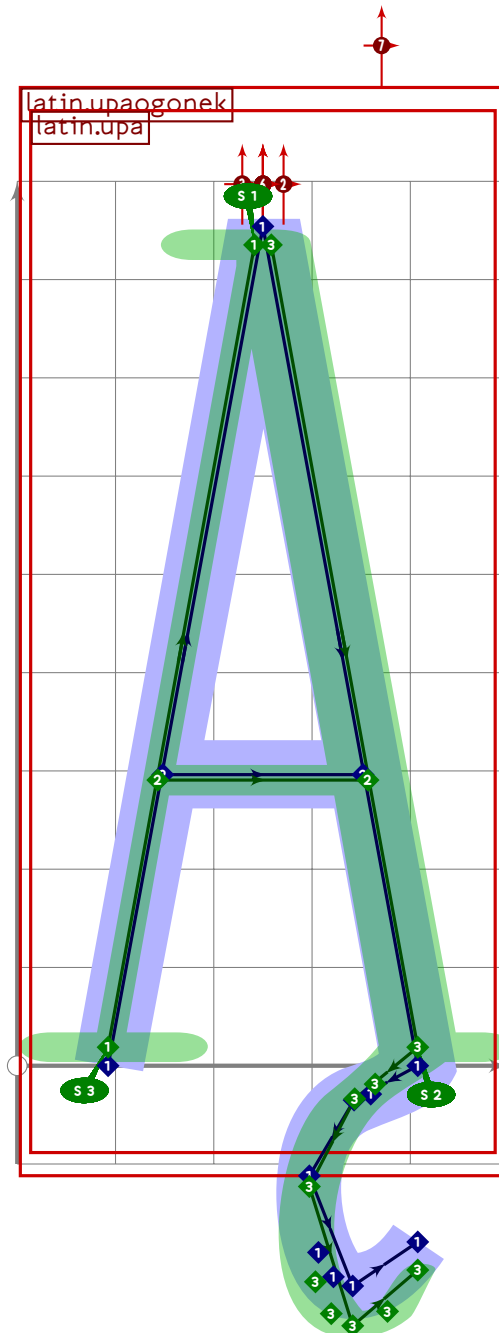
NUME

# ogonek.mp

```

1 %
2 % Ogonek letters for Tsukurimashou
3 % Copyright (C) 2011, 2012 Matthew Skala
4 %
5-29 [Standard copyright notice]
30
31 inclusion_lock(ogonek);
32
33

```

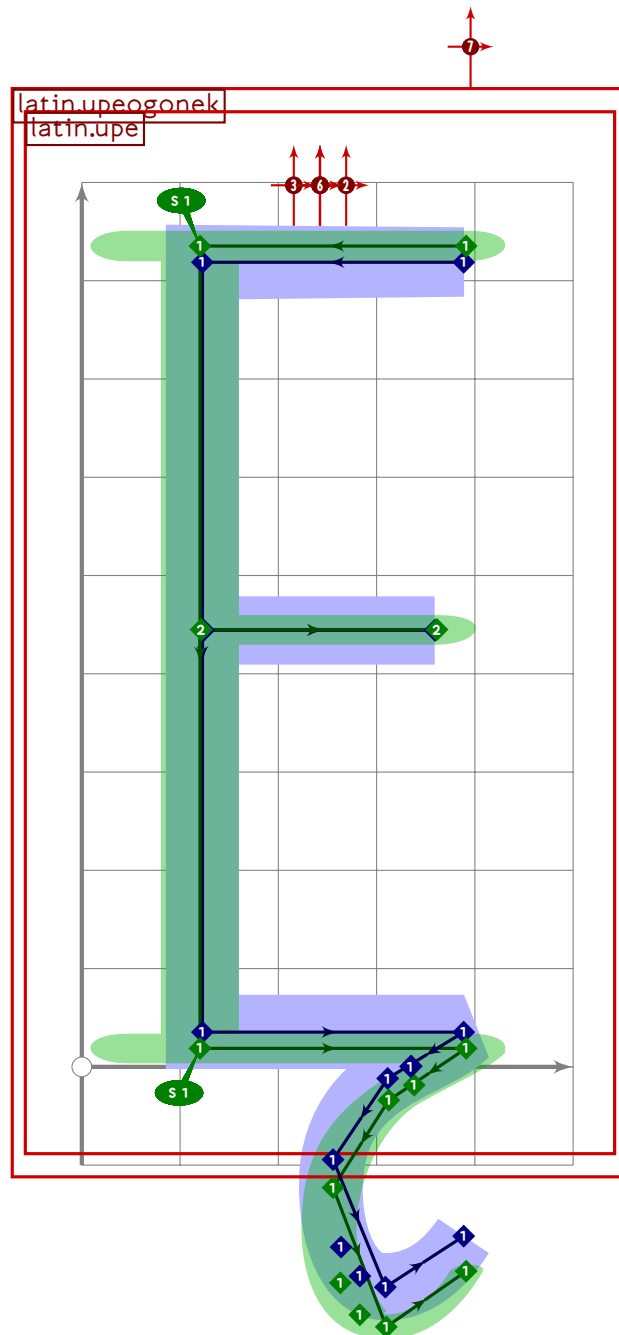


OGON

```

35 vardef latin.upaogonek =
36   push_pbox_toexpand("latin.upaogonek");
37   latin.upa;
38
39   x6=0.3[x2,x3];
40   x7=0.4[x6,x8];
41   x8=x3;
42
43   y6=0.5[y7,y3];
44   y7=latin_wide_desc_r;
45   y8=0.2[y7,y3];
46
47   if do_alteration:
48     replace_strokep(0)(oldp{dir 210}..z6..z7{right}..z8);
49     replace_strokep(0)(insert_nodes(oldp)(length(oldp)-2.5));
50     replace_strokeq(0)(oldq-(1.4,1.4)-(1.3,1.3)-(1.4,1.4)-(1,1));
51     set_boserif(0,1,2);
52     set_botip(0,length(get_strokep(0))-4,1);
53   else:
54     replace_strokep(-1)(oldp{dir 210}..z6..z7{right}..z8);
55     replace_strokep(-1)(insert_nodes(oldp)(length(oldp)-2.5));
56     replace_strokeq(-1)(oldq-(1.4,1.4)-(1.3,1.3)-(1.4,1.4)-(1,1));
57     set_boserif(-1,1,2);
58     set_botip(-1,length(get_strokep(-1))-4,1);
59   fi;
60   expand_pbox;
61 enddef;

```



```

62
63 vardef latin.upeogonek =
64   push_pbox__toexpand("latin.upeogonek");
65   latin.upe;
66
67   x7=0.5[x3,x4];
68   x8=0.4[x7,x9];
69   x9=x4;
70
71   y7=0.5[y8,y4];
72   y8=latin_wide_desc_r;
73   y9=0.2[y8,y4];

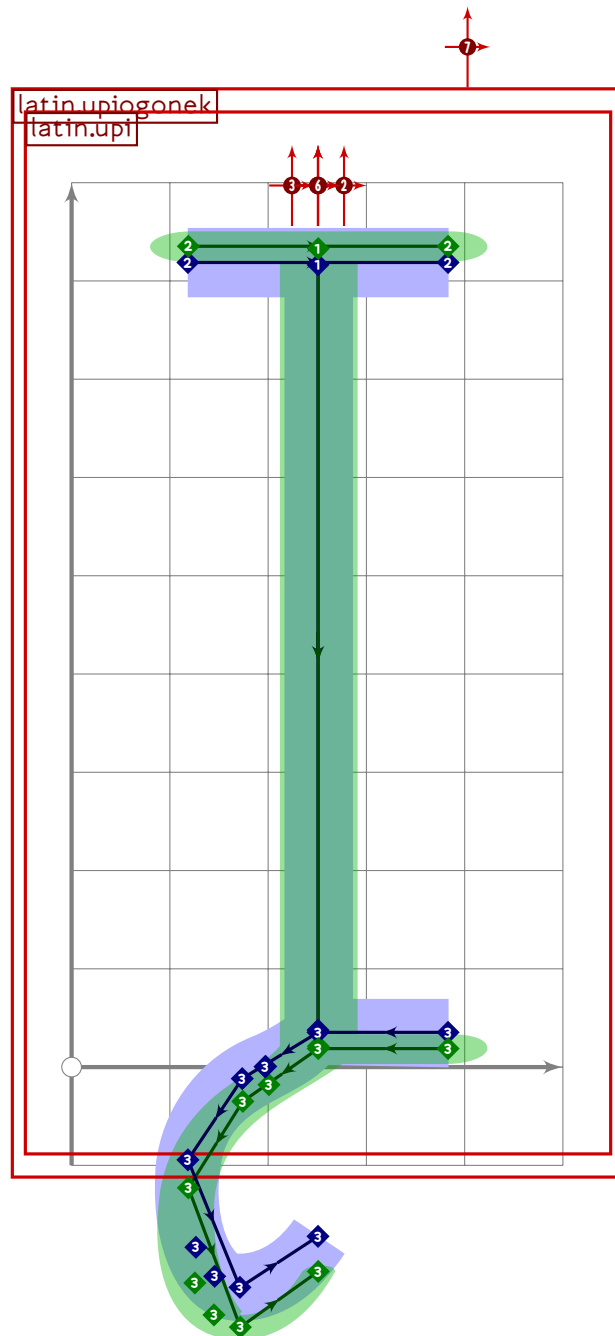
```

U+012E  
tsuku.Iogonek

```

74
75  replace_stroke(-1)(oldp{dir 210}..z7..z8{right}..z9);
76  replace_stroke(-1)(insert_nodes(oldp)(length(oldp)-2.5));
77  replace_strokeq(-1)(oldq-(1,4,1,4)-(1,3,1,3)-(1,4,1,4)-(1,1));
78  set_botip(-1,length(get_stroke(-1))-4,0);
79  expand_pbox;
80 enddef;

```



OGON

```

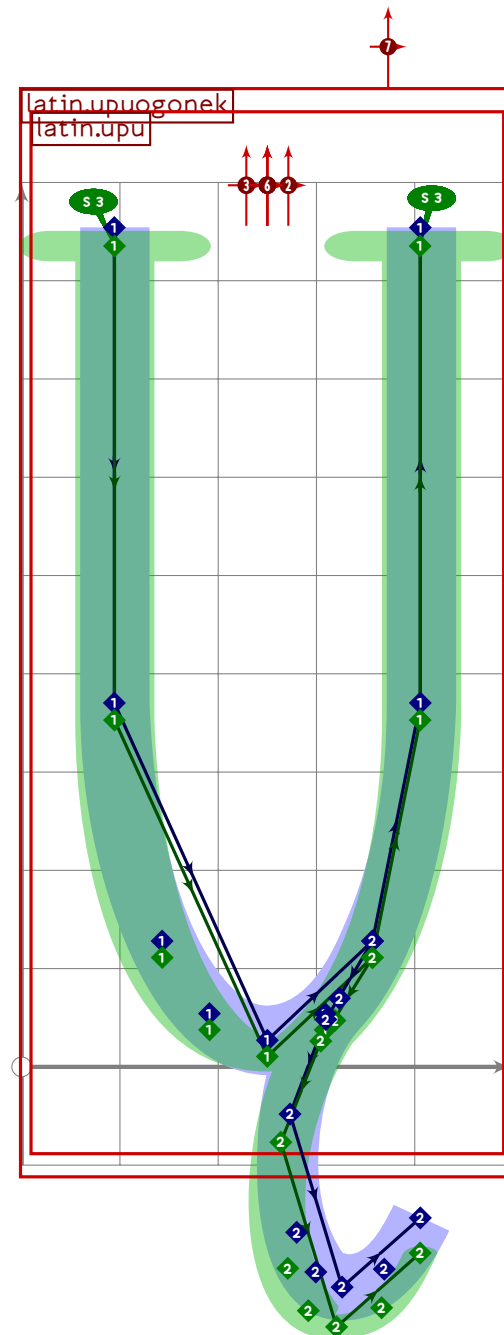
81
82 vardef latin.upiogonek =
83   push_pbox_toexpand("latin.upiogonek");
84   latin.upi;
85

```

```

86 x1=x4=500;
87 x2=300;
88 x3=0.4[x2,x4];
89
90 y1=latin_wide_low_h;
91 y2=0.5[y3,y1];
92 y3=latin_wide_desc_r;
93 y4=0.2[y3,y1];
94
95 replace_strokep(0)((700,latin_wide_low_h)-z1{dir 210}..z2..z3{right}..z4);
96 replace_strokep(0)(insert_nodes(oldp)(1.5));
97 replace_strokeq(0)((1.6,1.6)-(1.6,1.6)-(1.4,1.4)-
98   (1.3,1.3)-(1.4,1.4)-(1,1));
99 expand_pbox;
100 endif;

```



```

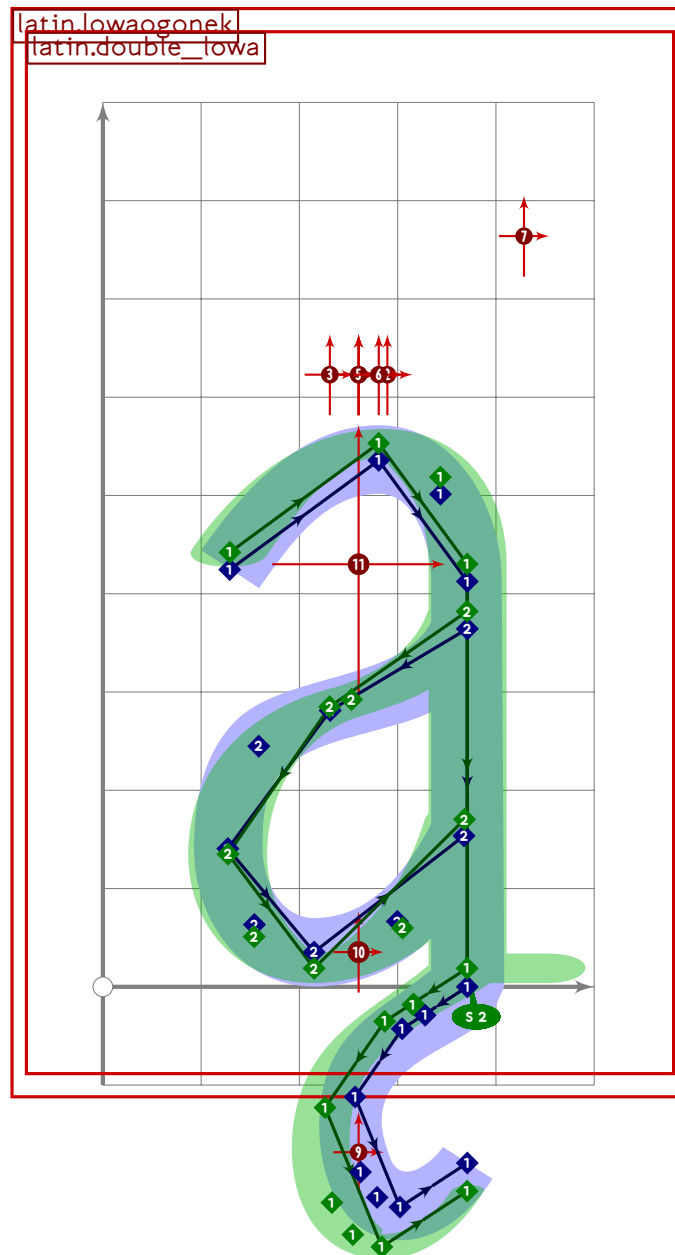
101
102 vardef latin.upuogonek =
103   push_pbox_toexpand("latin.upuogonek");
104   latin.upu;
105
106   replace_strokep(0)(insert_nodes(oldp)(2,5));
107   replace_strokeq(0)(insert_nodes(oldq)(2,5));
108   set_boserif(0,5,3);
109   set_boserif(0,4,whatever);
110
111   z6=point 3 of get_strokep(0);
112

```

```

113 y7=0.5[y8,y6];
114 y8=latin_wide_desc_r;
115 y9=0.2[y8,y6];
116
117 x9-x7=(x4-x2)*((y6-y8)/(y1-y3));
118 x8=0.4[x7,x9];
119 x9=x4;
120
121 push_stroke(z6{direction 3 of get_stroke(0)}..z7..z8{right}..z9,
122   (1,1,1)-(14,14)-(1.3,1.3)-(14,14)-(1,1));
123 replace_stroke(0,insert_nodes(oldp)(length(oldp)-2.5));
124 expand_pbox;
125 enddef;

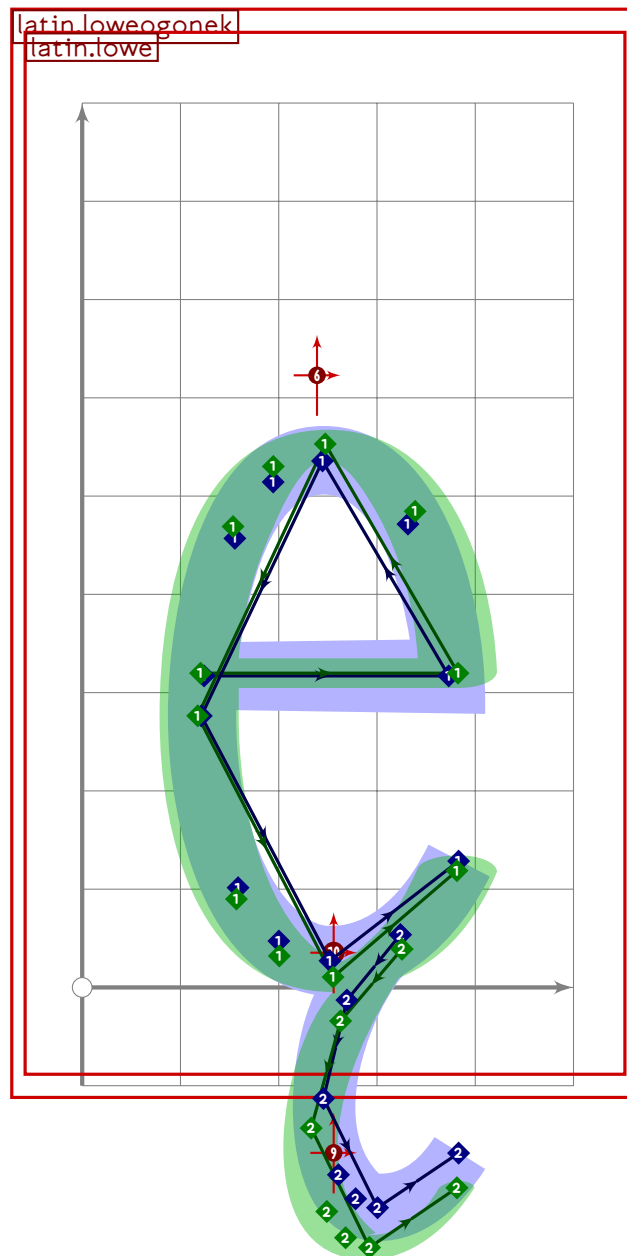
```



```

127 vardef latin.lowaogonek =
128   push_pbox_toexpand("latin.lowaogonek");
129   latin.lowa;
130
131   y9=0.5[y10,y4];
132   y10=latin_wide_desc_r;
133   y11=0.2[y10,y4];
134
135   x11-x9=(x4-x1)*((y4-y10)/(y2-y4));
136   x10=0.4[x9,x11];
137   x11=x4;
138
139   replace_strokep(-1)(oldp{dir 210}..z9..z10{right}..z11);
140   replace_strokep(-1)(insert_nodes(oldp)(length(oldp)-2.5));
141   replace_strokeq(-1)(oldq-(1.4,1.4)-(1.3,1.3)-(1.4,1.4)-(1,1));
142   set_botip(-1,length(get_strokep(-1))-4,1);
143   expand_pbox;
144 enddef;

```



```

145
146 vardef latin.loweogonek =
147   push_pbox_toexpand("latin.loweogonek");
148   latin.lowe;
149
150   z7=point (-0.5*length get_strokep(0)) of get_strokep(0);
151
152   y8=0.4[y9,y7];
153   y9=latin_wide_desc_r;
154   y10=0.2[y9,y7];
155
156   x10-x8=(x2-x4)*((y7-y9)/(y3-y5));
157   x9=0.4[x8,x10];
158   x10=x6;

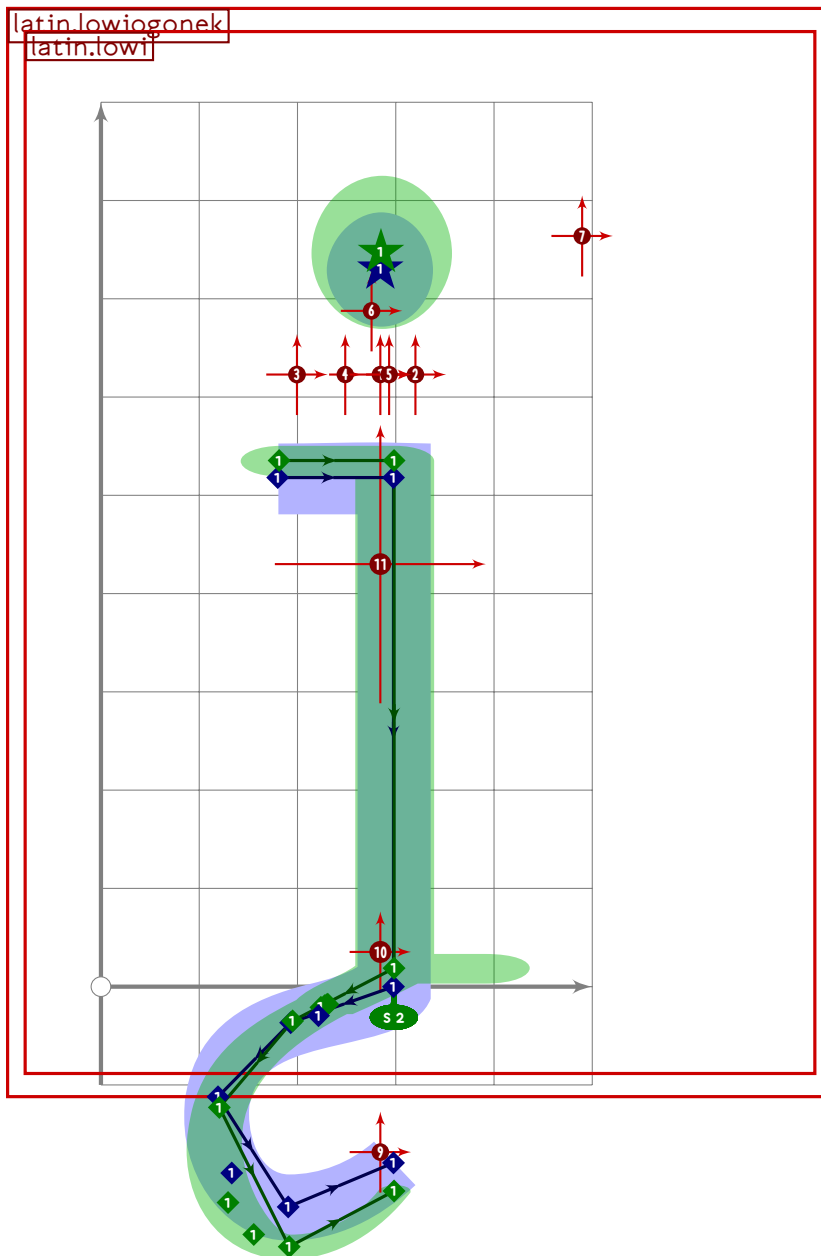
```

U+012F  
tsuku.iogonek

```

159
160 push_stroke(z7{direction (-0.5+length get_stroke(0)) of get_stroke(0)}
161 ..z8..z9{right}..z10,
162 (1.1,1.1)-(14,14)-(1.3,1.3)-(14,14)-(1,1));
163 replace_stroke(0,insert_nodes(oldp)(length(oldp)-2.5));
164 expand_pbox;
165 enddef;

```



```

166
167 vardef latin.lowiogonek =
168 push_pbox_toexpand("latin.lowiogonek");
169 latin.lowi;
170
171 x5=x7-200;
172 x6=0.4[x5,x7];

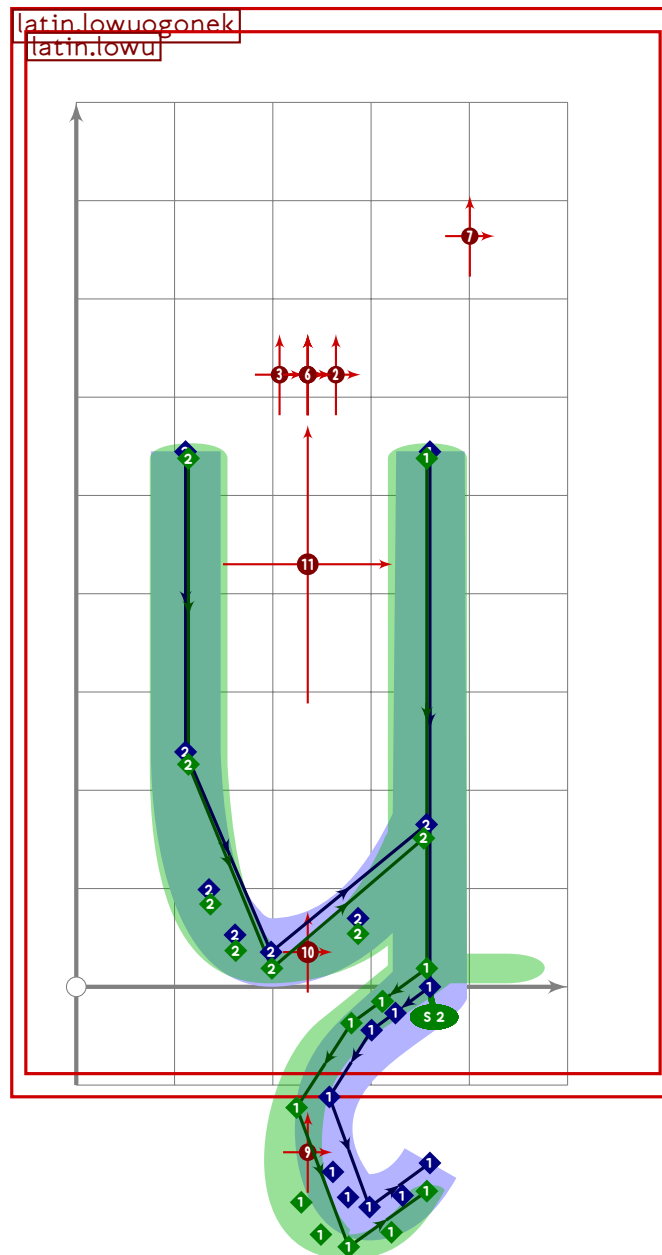
```

OGON

```

173 x7=x3;
174
175 y5=0.5[y6,y3];
176 y6=latin_wide_desc_r;
177 y7=0.2[y6,y3];
178
179 replace_strokep(0)(oldp{dir 210}.z5..z6{right}.z7);
180 replace_strokep(0)(insert_nodes(oldp)(2.5));
181 replace_strokeq(0)(oldq-(1.4,1.4)-(1.3,1.3)-(1.4,1.4)-(1,1));
182 set_boserif(0,2,2);
183 set_botip(0,2,1);
184 expand_pbox;
185 enddef;

```



```

187 vardef latin.lowuogonek =
188   push_pbox_toexpand("latin.lowuogonek");
189   latin.lowu;
190
191   y7=0.5[y8,y1];
192   y8=latin_wide_desc_r;
193   y9=0.2[y8,y1];
194
195   x9-x7=(x1-x6)*((y1-y8)/(y2-y1));
196   x8=0.4[x7,x9];
197   x9=x1;
198
199   replace_strokep(-1)(z2-z1{dir 210}..z7..z8{right}..z9);
200   replace_strokep(-1)(insert_nodes(oldp)(length(oldp)-2.5));
201   replace_strokeq(-1)((1.6,1.6)-(1.6,1.6)-(1.4,1.4)-(1.3,1.3)-
202     (1.4,1.4)-(1,1));
203   set_botip(-1,1,1);
204   set_boserif(-1,0,whatever);
205   set_boserif(-1,1,2);
206   expand_pbox;
207 enddef;

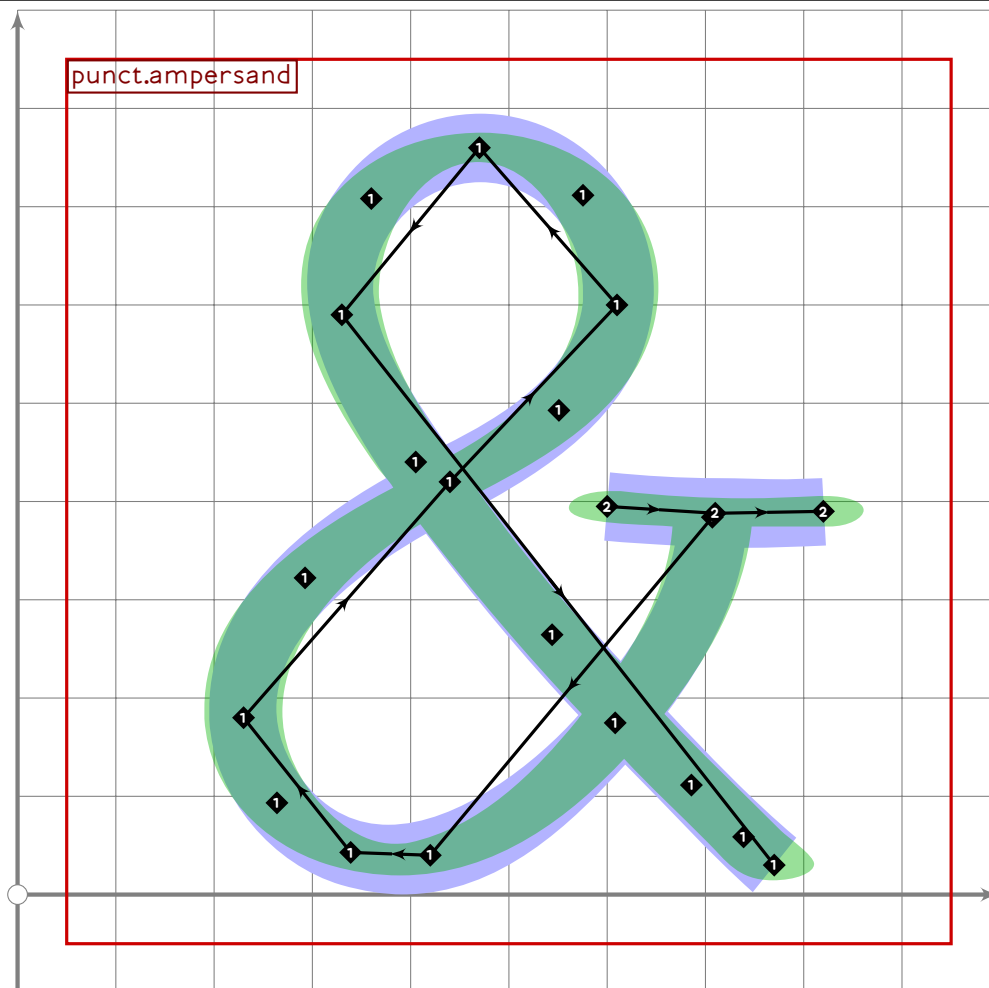
```

# punct.mp

```

1 %
2 % Punctuation for Tsukurimashou
3 % Copyright (C) 2011 Matthew Skala
4 %
5-29 [Standard copyright notice]
30
31 inclusion_lock(punct);
32
33

```



```

34
35 vardef punct.ampersand =
36   push_pbox_toexpand("punct.ampersand");
37
38   push_stroke((707,384)..tension 1.3..(420,40)..(230,180)..(440,420)..
39     (610,600)..(470,760)..(330,590)..tension 1.5 and 4..(770,30),
40     (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-
41     (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
42   replace_strokep(0)(insert_nodes(oldp)(1.3));
43

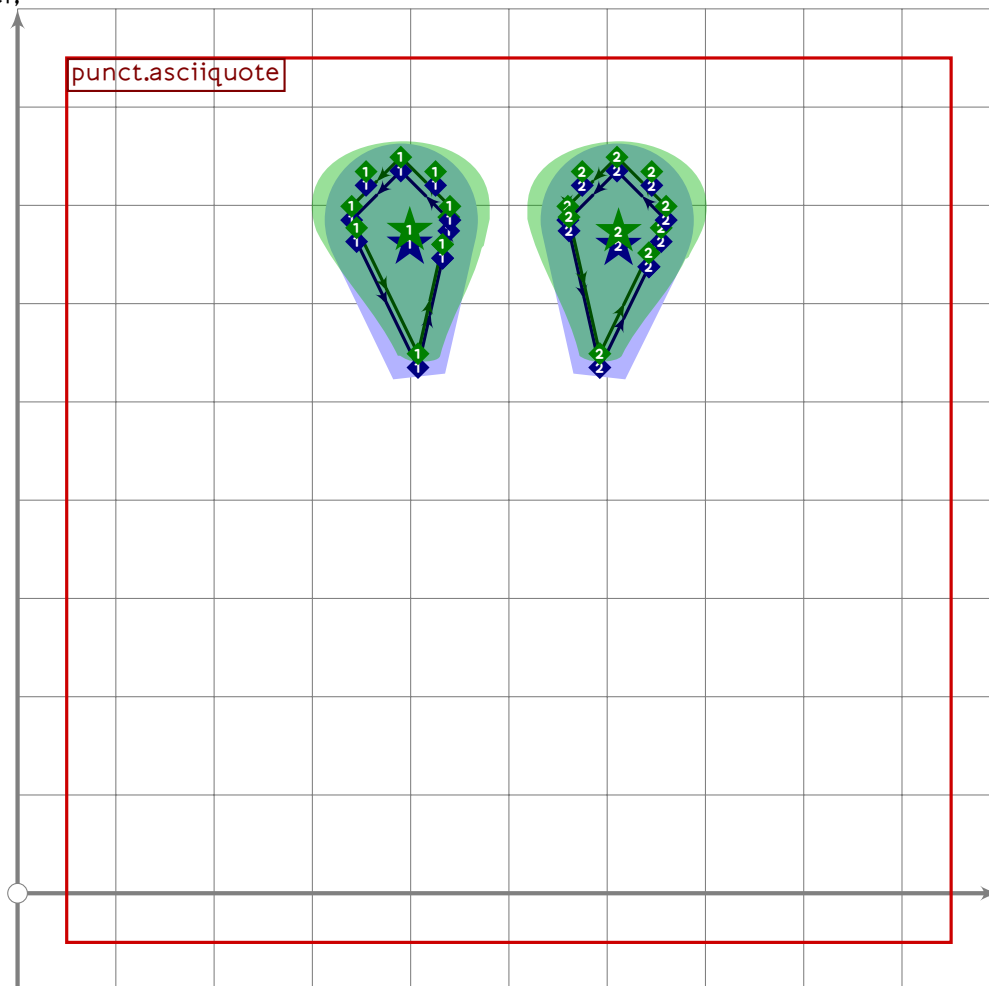
```

U+FF02  
tsuku.uniFF02

```

44  push_stroke((600,395)..(710,388)..(820,390),
45    (1.6,1.6)-(1.6,1.6)-(1.6,1.6));
46  expand_pbox;
47  enddef;

```



```

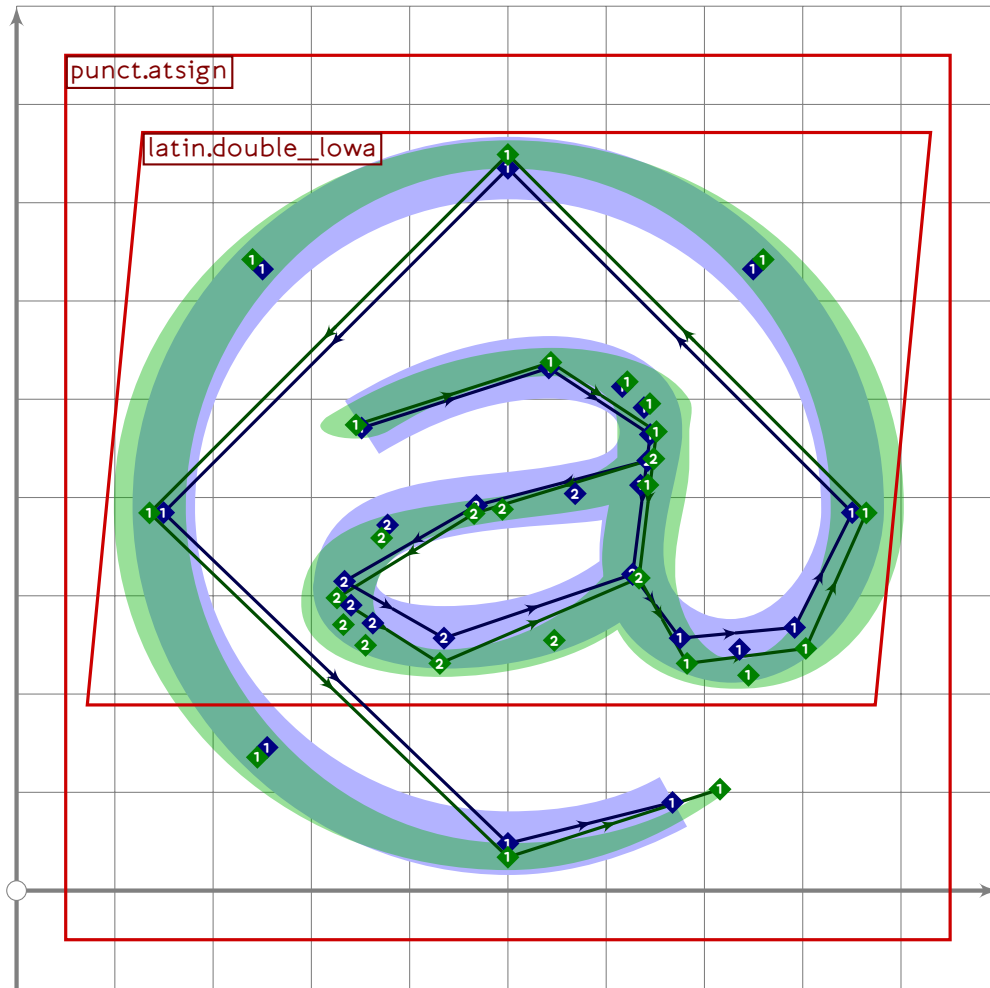
48
49  vardef punct.asciquote =
50    push_pbox_toexpand("punct.asciquote");
51
52    numeric dx;
53    dx=tsu_punct_size;
54
55    (x1+x2)/2=(x3+x4)/2=500;
56    x2-x1=1.2*dx*tsu_punct_size;
57    x4-x3=tsu_punct_size*1.85;
58
59    y1=y2=latin_wide_high_r-dx/2;
60    y3=y4=y1-1.5*dx;
61
62    path ptmp;
63    ptmp:=(down..right..up..left..cycle)
64      scaled (abs(z3-z1)+-(dx/2));

```

```

65
66 push_stroke((down..right..up..left..cycle) scaled (dx/2) shifted z1,
67   (2,2)-(2,2)-(2,2)-(2,2)-(2,2)-(2,2)-(1.3,1.3)-cycle);
68 replace_strokep(0)(z3-(subpath ((xpart (oldp intersectiontimes
69   (ptmp shifted z3))),4-xpart ((reverse oldp) intersectiontimes
70   (ptmp shifted z3)))) of oldp)-cycle);
71 replace_strokep(0)((subpath (0.8,6) of oldp)-cycle);
72 set_bosize(0,75);
73 set_botip(0,6,0);;
74
75 push_stroke((down..right..up..left..cycle) scaled (dx/2) shifted z2,
76   (2,2)-(2,2)-(2,2)-(2,2)-(2,2)-(2,2)-(1.3,1.3)-cycle);
77 replace_strokep(0)(z4-(subpath ((xpart (oldp intersectiontimes
78   (ptmp shifted z4))),4-xpart ((reverse oldp) intersectiontimes
79   (ptmp shifted z4)))) of oldp)-cycle);
80 replace_strokep(0)((subpath (0.8,6) of oldp)-cycle);
81 set_bosize(0,75);
82 set_botip(0,6,0);;
83
84 if tsu_pbrush_size>=30:
85   push_lcblob(get_strokep(-1));
86   push_lcblob(get_strokep(0));
87 fi;
88 expand_pbox;
89 endif;

```



```

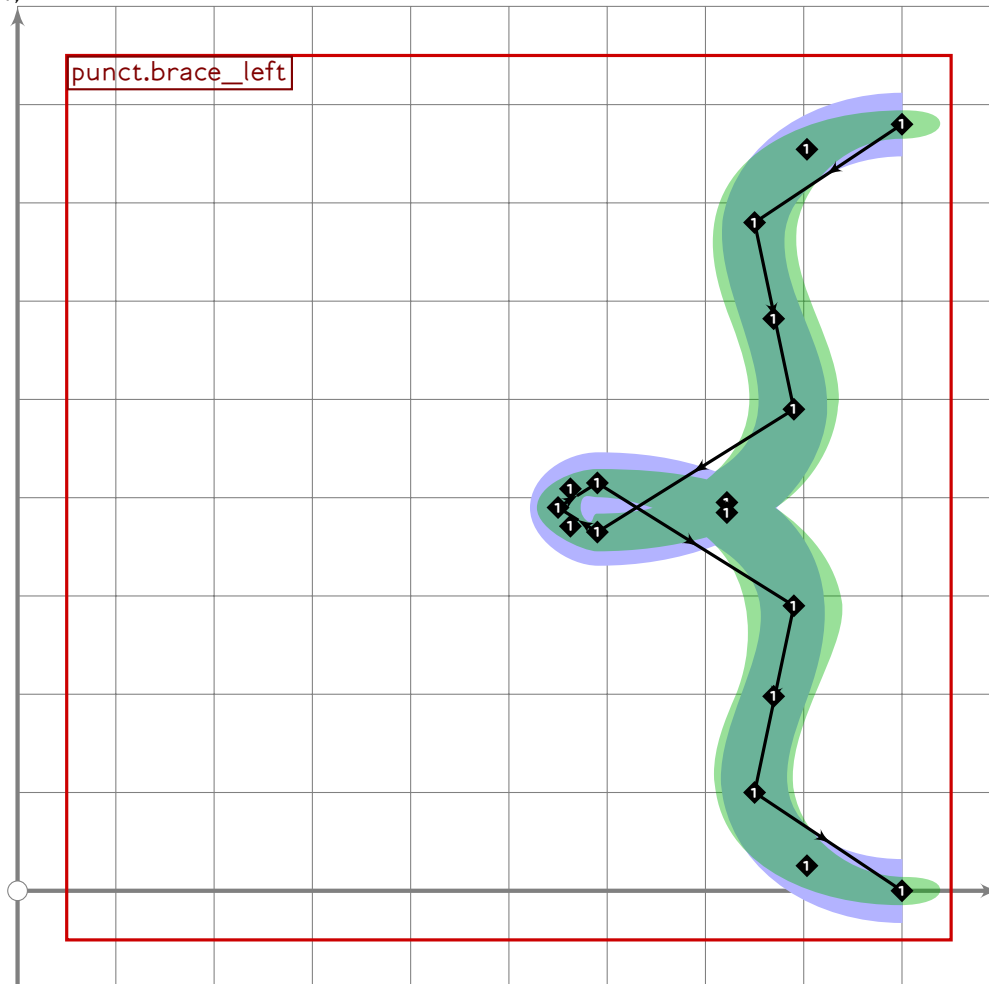
90
91 vardef punct.atsign =
92   push_pbox_toexpand("punct.atsign");
93   begingroup
94     save xsp,ysp;
95     xsp:=sp;
96     latin.low_a;
97     set_boserif(-1,3,whatever);
98     ysp:=sp;
99
100    numeric x[],y[];
101    x1-x2=x2-x3=y2-y1;
102    x2=x4=500;
103    y1=y3=0.49[y4,y2];
104    y2=latin_wide_high_r;
105    y4=latin_wide_low_r;
106
107    transform shrinka;
108    (0.5[lcorner get_stroke(-1),urcorner get_stroke(-1)])
109      transformed shrinka=0.5[z3,z1];
110    (0.5[lcorner get_stroke(-1),urcorner get_stroke(-1)])

```

```

111   transformed shrinka=0.71[z3,z1];
112   (0.5[ulcorner get_stroke(-1),urcorner get_stroke(-1)])
113   transformed shrinka=z2+(0.07;1)*0.29*(x1-x3);
114   sp:=xsp;
115   tsu_xform(shrinka shifted (-10,0))(sp:=ysp);
116
117   z5=point infinity of get_stroke(0);
118   y6=ypart lrcorner get_stroke(0);
119   x6=0.5[x2,x1];
120   replace_stroke(-1)((subpath (0,length(oldp)-1) of oldp)..z5..z6..
121     (subpath (0,3.85) of (z1..z2..z3..z4..cycle)));
122   replace_stroke(-1)(insert_nodes(oldp)((length oldp-4.5)));
123   replace_stroke(-1)(oldq-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-
124     (1.6,1.6)-(1.6,1.6)-(0,0));
125   endgroup;
126   expand_pbox;
127   enddef;

```



```

128
129 vardef punct.brace_left =
130   push_pbox_toexpand("punct.brace_left");
131   push_stroke((900,780){left}..

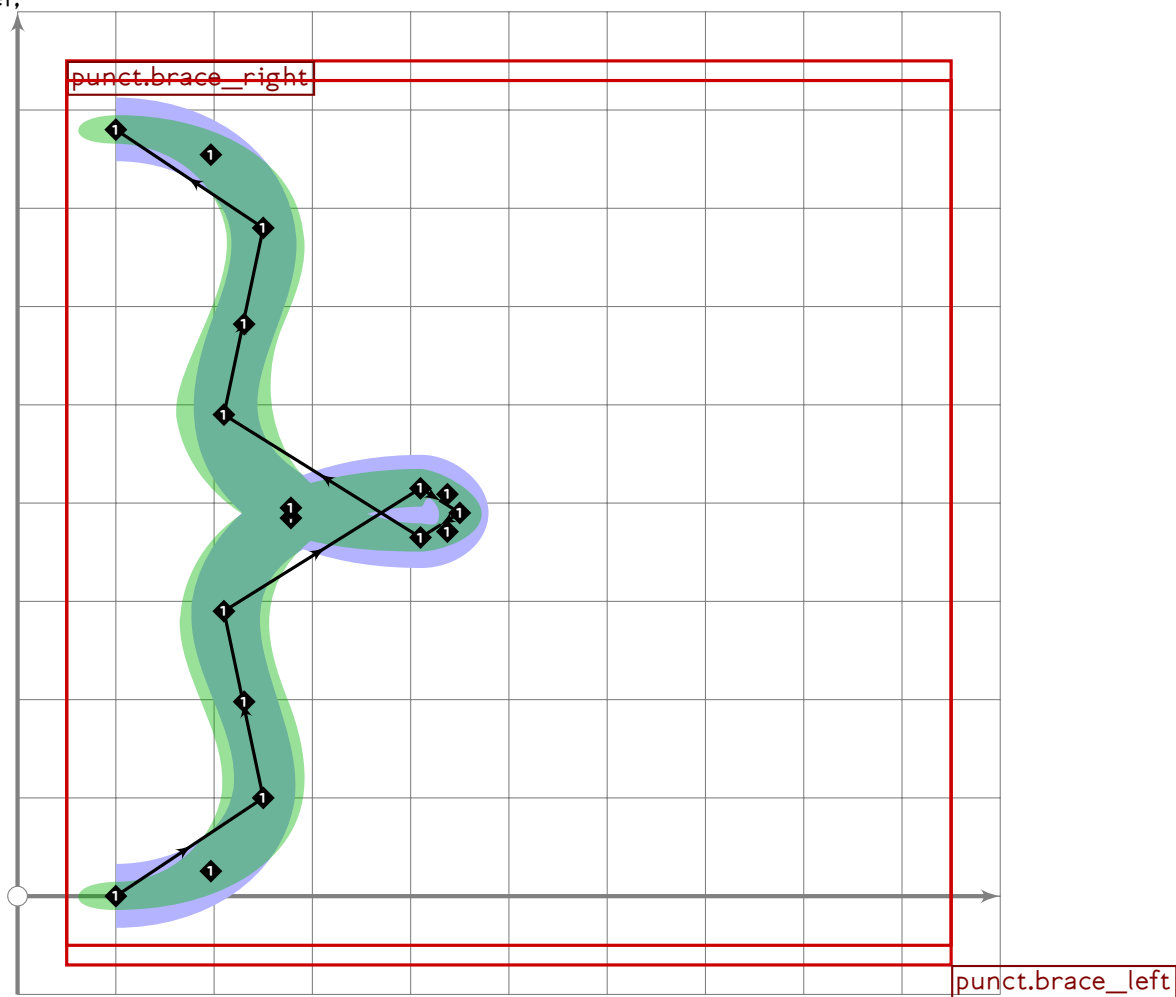
```

U+FF5D  
tsuku.uniFF5D

```

132 (900-1.5*tsu_punct_size,680)..
133 (900-1.1*tsu_punct_size,490)..
134 (900-3.1*tsu_punct_size,390-0.25*tsu_punct_size){left}..
135 (900-3.5*tsu_punct_size,390)..
136 (900-3.1*tsu_punct_size,390+0.25*tsu_punct_size){right}..
137 (900-1.1*tsu_punct_size,290)..
138 (900-1.5*tsu_punct_size,100)..
139 (900,0){right},
140 (1.7,1.7)-(2,2)-(2,2)-
141 (1.2,1.2)-(1.2,1.2)-(1.2,1.2)-
142 (2,2)-(2,2)-(1.7,1.7));
143 set_bosize(0,90);
144 expand_pbox;
145 enddef;

```

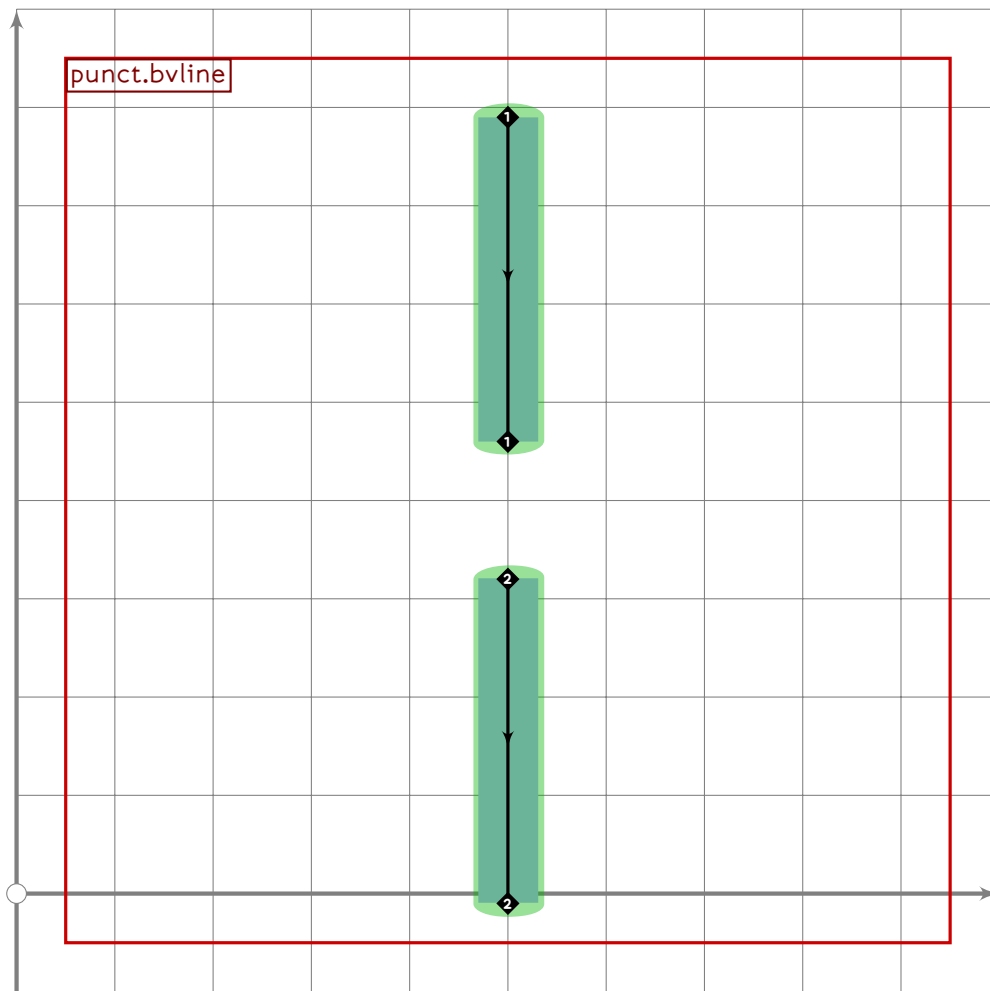


```

146 vardef punct.brace_right =
147   push_pbox_toexpand("punct.brace_right");
148   tsu_xform(identity rotatedaround (centre_pt,180))
149   (punct.brace_left);
150   expand_pbox;
151 enddef;

```

**PUNC**



```

152
153 vardef punct.bvline =
154   push_pbox__toexpand("punct.bvline");
155
156   push_stroke((500,690+tsu_punct_size)-(500,390+0.7*tsu_punct_size),
157     (1.6,1.6)-(1.6,1.6));
158   set_bosize(0,90);
159
160   push_stroke((500,390-0.7*tsu_punct_size)-(500,90-tsu_punct_size),
161     (1.6,1.6)-(1.6,1.6));
162   set_bosize(0,90);
163   expand_pbox;
164 enddef;
165
166 vardef punct.make_comma(expr cpos,cang) =
167   begingroup
168     save x,y,t,u,xsp;
169     numeric x[],y[];
170     transform t,u;
171     xsp:=sp;
172     sp:=1;

```

```

173 t:=tsu_rescale_xform;
174 sp:=xsp;
175
176 x1=0.8[x2,x4];
177 (x2+x4)/2=x3=0;
178 (x2-x4)=0.45*(y3-y1)=tsu_punct_size;
179 x5=x3;
180
181 y2=y4=0.32[y3,y1]=0;
182 y5-y4=0.73*(y4-y3);
183
184 push_stroke(z1{curl 0.2}..tension 1.2..z2..z3..z4{dir 280}..
185   z5.{curl 0.2}(point 0.8 of (z1{curl 0.2}..tension 1.2..
186   z2..z3..z4{dir 280})),
187   (2,2)-(1,1)-(2,2)-(2,2)-(2,2)-(2,2)-(2,2));
188 replace_strokep(0)((point 4.2 of oldp)-oldp);
189 (0,0) transformed u=llcorner get_strokep(0);
190 (1,0) transformed u=lrcorner get_strokep(0);
191 (0,1) transformed u=ulcorner get_strokep(0);
192 replace_strokep(0)(oldp rotated (cang-6) shifted (cpos transformed t)
193   transformed inverse t);
194 u:=u scaled 1.3 rotated (cang-6) shifted (cpo transformed t)
195   transformed inverse t;
196 set_botip(0,1,0);
197
198 if tsu_pbrush_size>=30:
199   replace_strokep(0)(subpath (0.03,5.75) of oldp);
200   set_bosize(0,40);
201   push_lcblob((subpath (1.6,4.9) of get_strokep(0))-cycle);
202 else:
203   replace_strokep(0)(subpath (0,5.2) of oldp);
204   set_bosize(0,80);
205 fi;
206 push_pbox_explicit("punct.make_comma",u);
207 endgroup;
208 endif;
209
210 vardef punct.make_revcomma(expr cpos,cang) =
211   begingroup
212     save x,y,t,u,xsp;
213     numeric x[],y[];
214     transform t,u;
215     xsp:=sp;
216     sp:=1;
217     t:=tsu_rescale_xform;
218     sp:=xsp;
219
220     x1=0.8[x2,x4];

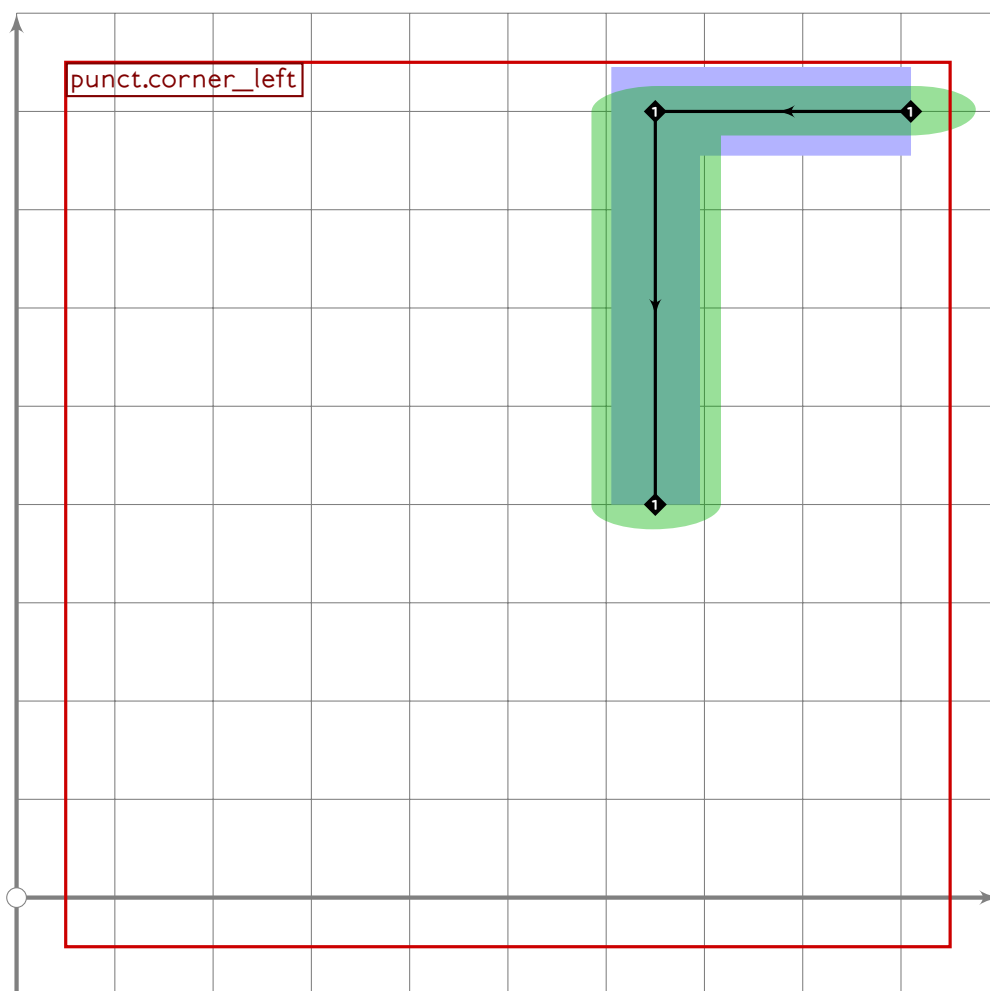
```

```

221 (x2+x4)/2=x3=0;
222 (x2-x4)=0.45*(y3-y1)=tsu_punct_size;
223 x5=x3;
224
225 y2=y4=0.32[y3,y1]=0;
226 y5-y4=0.73*(y4-y3);
227
228 push_stroke(z1{curl 0.2}..tension 1.2..z2..z3..z4{dir 280}..
229     z5.{curl 0.2}(point 0.8 of (z1{curl 0.2}..tension 1.2..
230     z2..z3..z4{dir 280})),
231     (2,2)-(1,1)-(2,2)-(2,2)-(2,2)-(2,2)-(2,2));
232 replace_strokep(0)((point 4.2 of oldp)-oldp);
233 (0,0) transformed u=llcorner get_strokep(0);
234 (1,0) transformed u=lrcorner get_strokep(0);
235 (0,1) transformed u=ulcorner get_strokep(0);
236 replace_strokep(0)(reverse (oldp rotated -6 reflectedabout(up,down)));
237 replace_strokep(0)(oldp rotated (cang-6) shifted (cpos transformed t)
238     transformed inverse t);
239 u:=u scaled 1.3 rotated (cang-6) shifted (cpo transformed t)
240     transformed inverse t;
241 set_botip(0,1,0);
242
243 if tsu_pbrush_size>=30:
244     replace_strokep(0)(subpath (0.25,5.97) of oldp);
245     set_bosize(0,40);
246     push_lcblob((subpath (1.1,4.4) of get_strokep(0))-cycle);
247 else:
248     replace_strokep(0)(subpath (0.8,6) of oldp);
249     set_bosize(0,80);
250 fi;
251 push_pbox_explicit("punct.make_revcomma",u);
252 endgroup;
253 enddef;

```

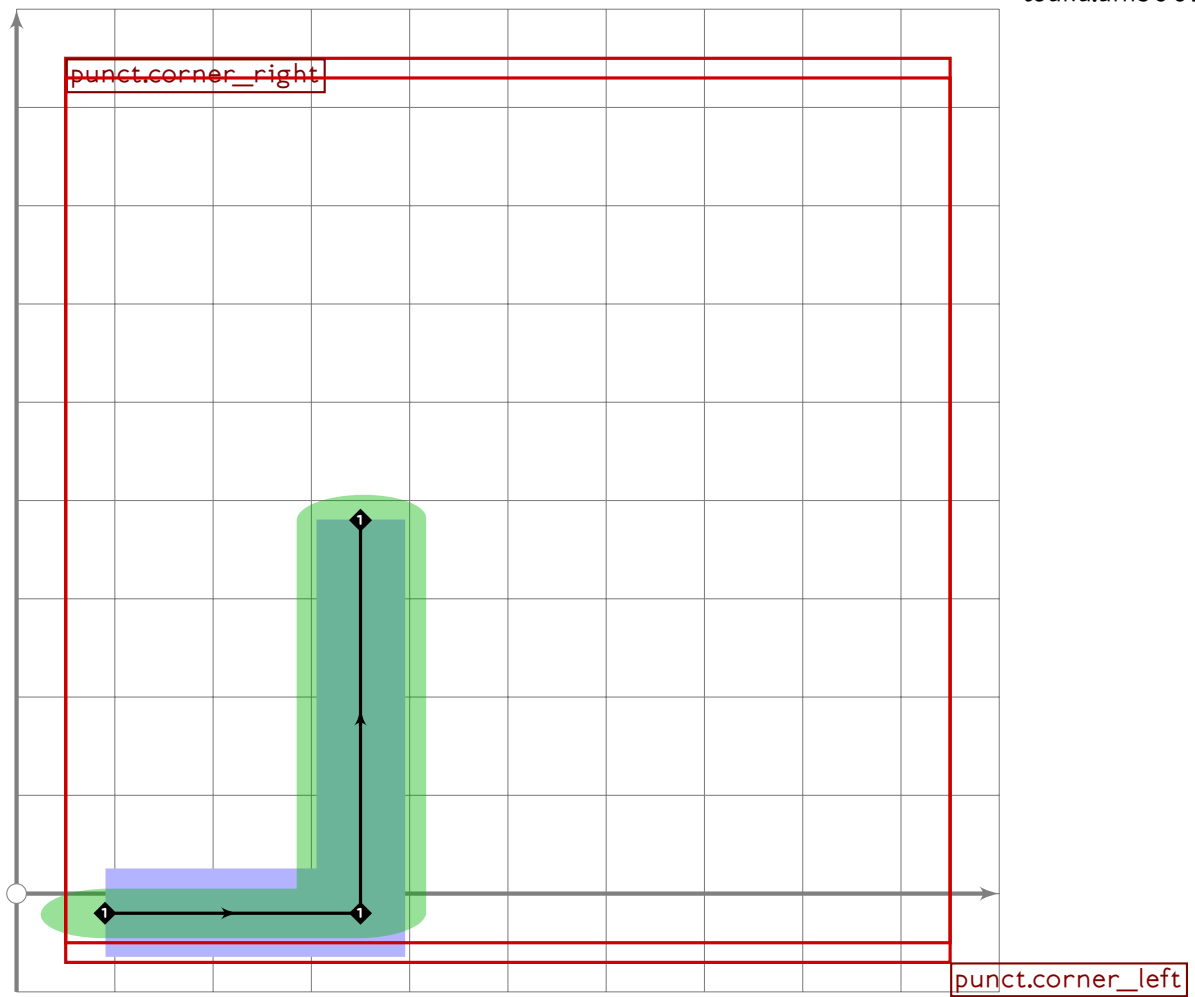
U+300C  
tsuku.uni300C



```

254
255 vardef punct.corner_left =
256   push_pbox_toexpand("punct.corner_left");
257   push_stroke((910,800)-(650,800)-(650,400),(2,2)-(2,2)-(2,2));
258   set_bosize(0,120);
259   set_botip(0,1,1);
260   expand_pbox;
261 enddef;

```

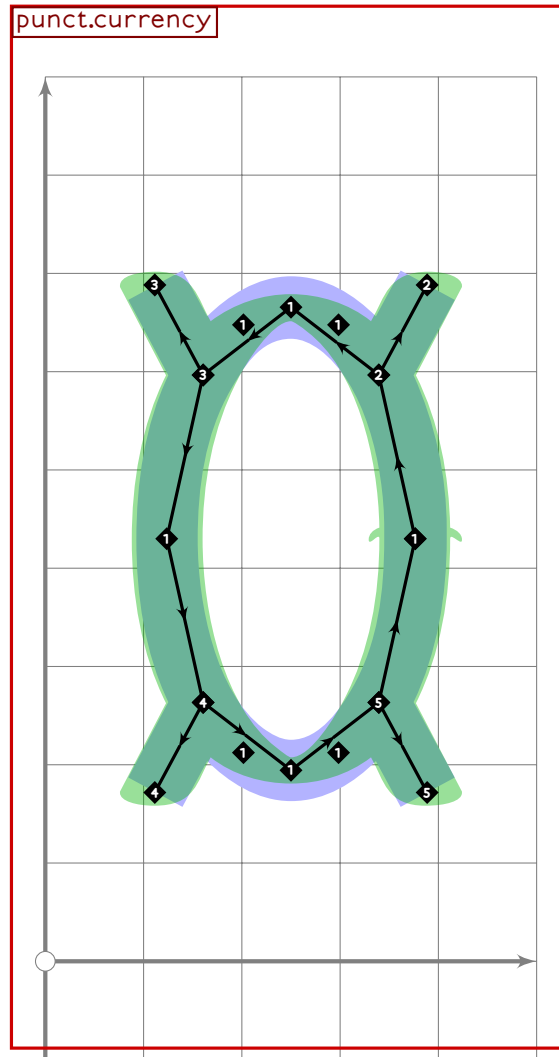


```

262 vardef punct.corner_right =
263   push_pbox_toexpand("punct.corner_right");
264   tsu_xform(identity rotatedaround (centre_pt,180))
265     (punct.corner_left);
266   expand_pbox;
267 enddef;

```

U+00A4  
tsuku.currency



```

268
269 vardef punct.currency =
270   push_pbox_toexpand("punct.currency");
271
272   push_stroke(fullcircle scaled (4*tsu_punct_size) shifted centre_pt,
273     (1.6,1.6)–(1.6,1.6)–(1.6,1.6)–(1.6,1.6)–cycle);
274   set_bosize(0,90);
275
276   push_stroke(((1,0)–(1.55,0)) rotated 45
277     scaled (2*tsu_punct_size) shifted centre_pt,
278     (1.6,1.6)–(1.6,1.6));
279   set_bosize(0,90);
280
281   push_stroke(((1,0)–(1.55,0)) rotated 135
282     scaled (2*tsu_punct_size) shifted centre_pt,
283     (1.6,1.6)–(1.6,1.6));
284   set_bosize(0,90);
285
286   push_stroke(((1,0)–(1.55,0)) rotated 225

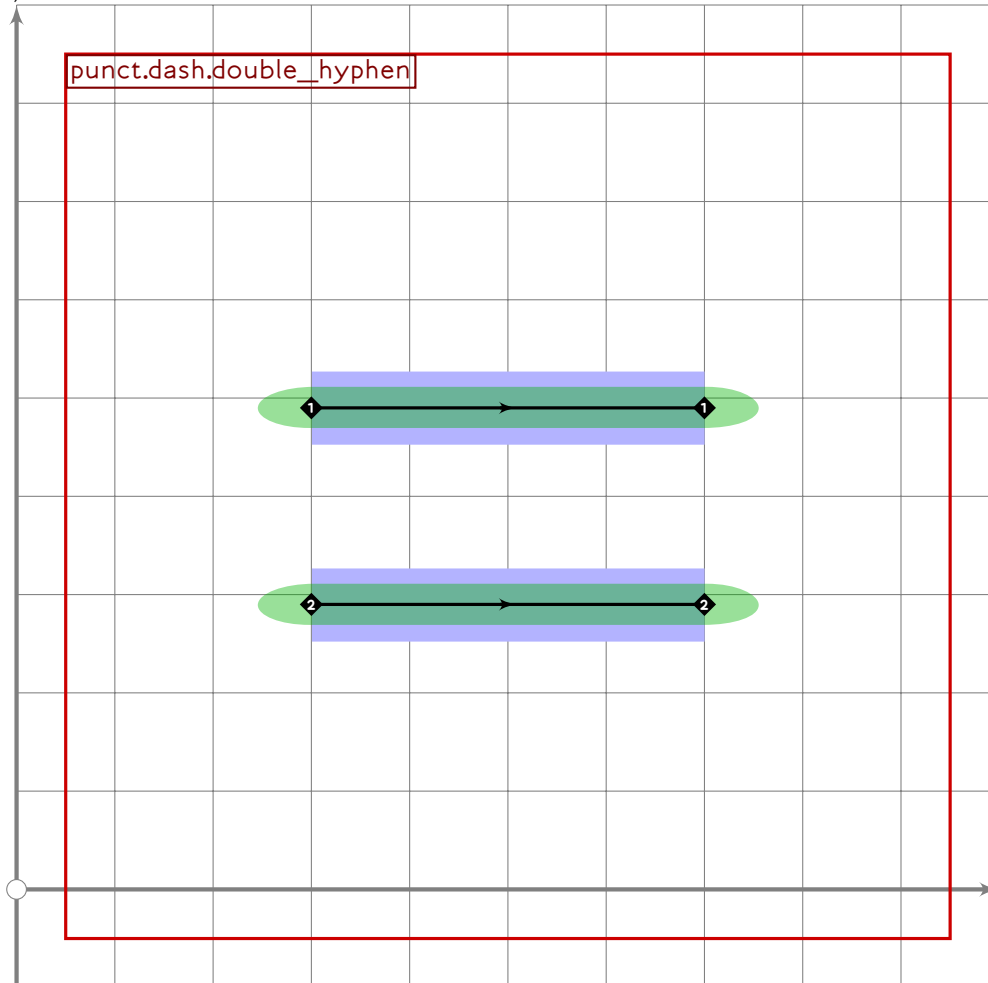
```

**PUNC**

```

287     scaled (2*tsu_punct_size) shifted centre_pt,
288     (1.6,1.6)-(1.6,1.6));
289     set_bosize(0,90);
290
291     push_stroke(((1,0)-(1.55,0)) rotated 315
292     scaled (2*tsu_punct_size) shifted centre_pt,
293     (1.6,1.6)-(1.6,1.6));
294     set_bosize(0,90);
295     expand_pbox;
296 enddef;

```



```

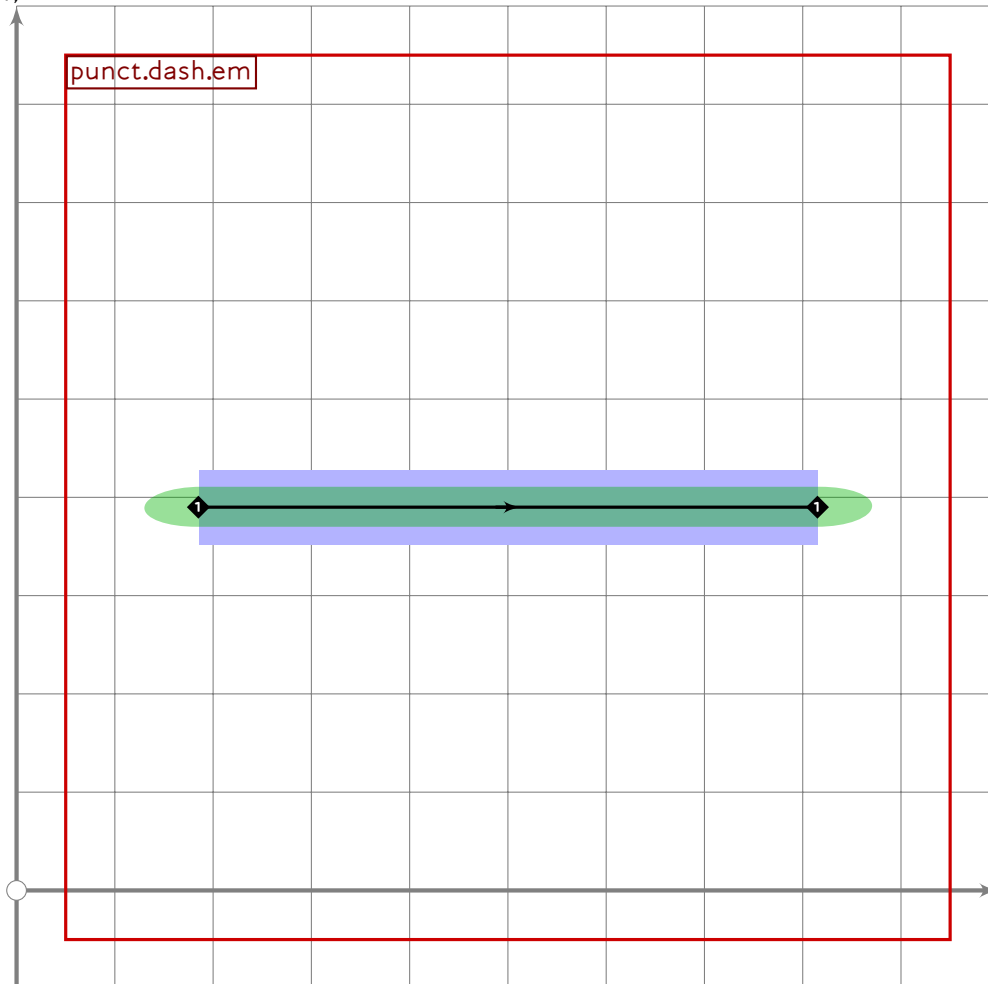
297
298 vardef punct.dash.double_hyphen =
299     push_pbox_toexpand("punct.dash.double_hyphen");
300
301     (z1+z4)/2=centre_pt;
302     x2-x1=400;
303     y1-y3=200;
304     y1=y2;
305     y3=y4;
306     x1=x3;
307     x2=x4;

```

```

308
309 push_stroke(z1-z2,(2,2)-(2,2));
310 push_stroke(z3-z4,(2,2)-(2,2));
311 expand_pbox;
312 enddef;

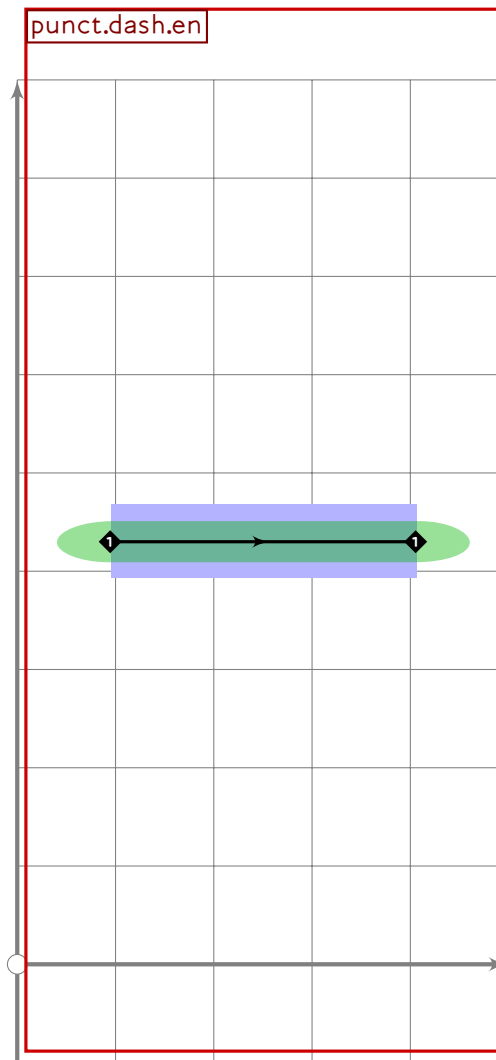
```



```

313
314 vardef punct.dash.em =
315   push_pbox_toexpand("punct.dash.em");
316   (z1+z2)/2=centre_pt;
317   x2-x1=if is_proportional: 750 else: 630 fi;
318   y1=y2;
319   push_stroke(z1-z2,(2,2)-(2,2));
320   expand_pbox;
321 enddef;

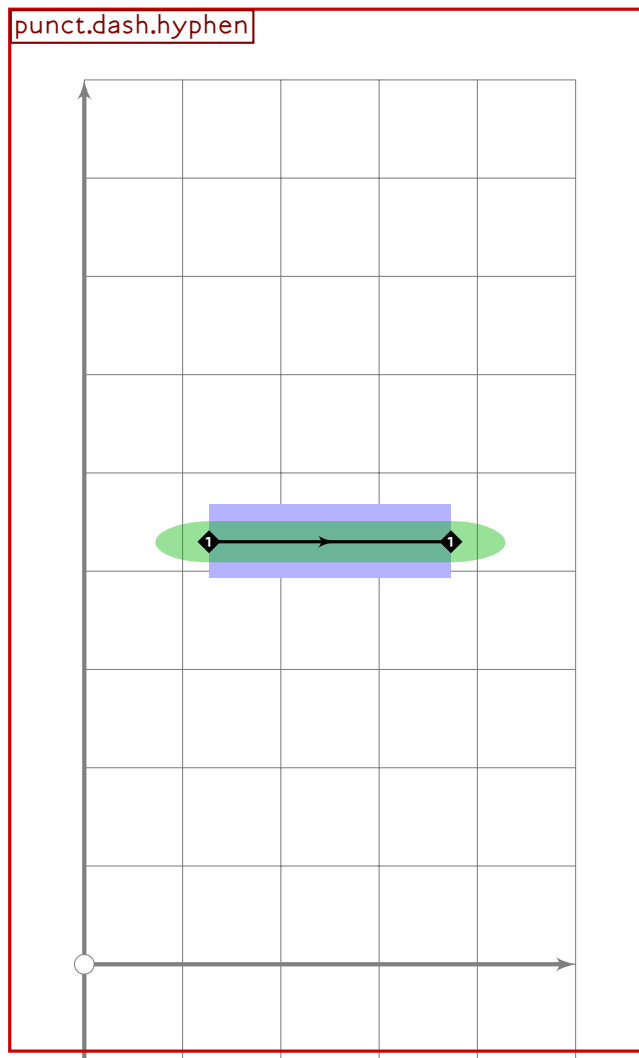
```



```

322
323 vardef punct.dash.en =
324   push_pbox_toexpand("punct.dash.en");
325   (z1+z2)/2=centre_pt;
326   x2-x1=580;
327   y1=y2;
328   push_stroke(z1-z2,(2,2)-(2,2));
329   expand_pbox;
330 enddef;

```



```

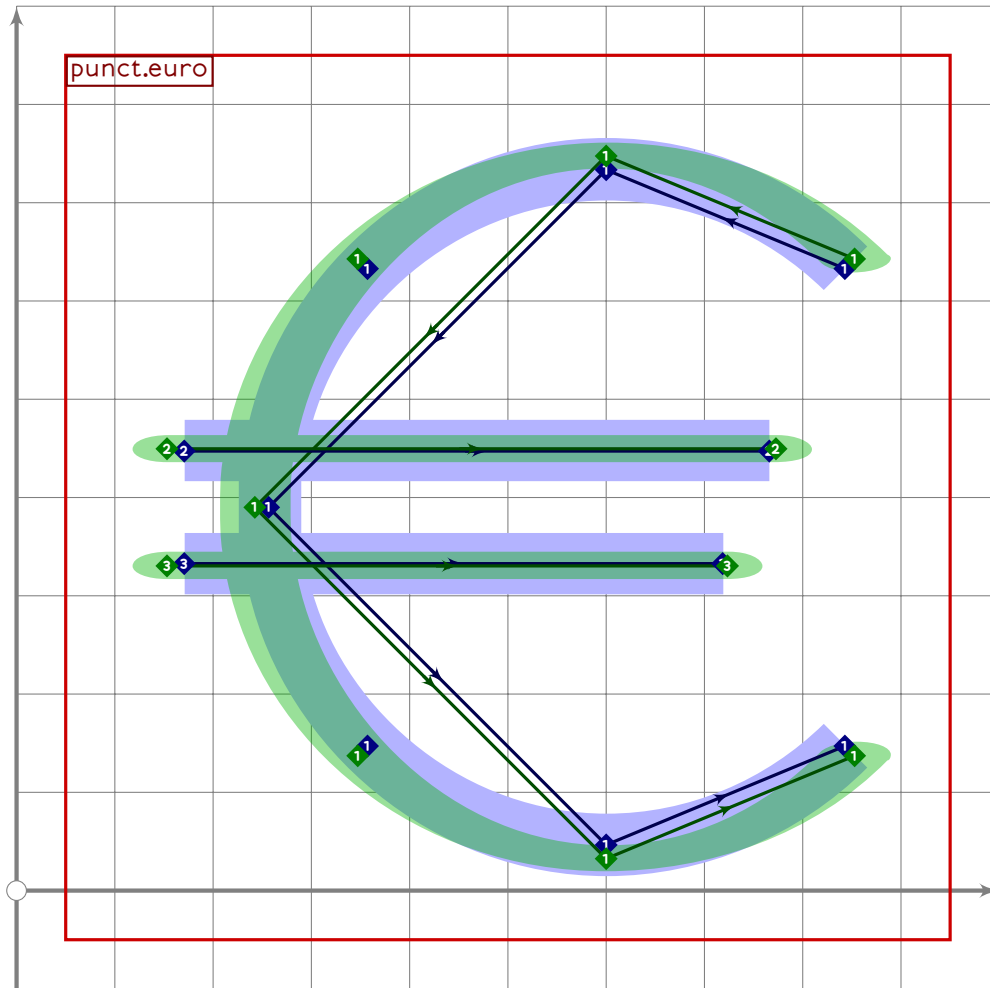
331
332 vardef punct.dash.hyphen =
333   push_pbox_toexpand("punct.dash.hyphen");
334   (z1+z2)/2=centre_pt;
335   x2-x1=340;
336   y1=y2;
337   push_stroke(z1-z2,(2,2)-(2,2));
338   expand_pbox;
339 enddef;
340
341 vardef punct.dash.long =
342   push_pbox_toexpand("punct.dash.long");
343   (z1+z2)/2=centre_pt;
344   x2-x1=340;
345   y1=y2;
346   push_stroke(z1-z2,(2,2)-(2,2));
347   expand_pbox;
348 enddef;
349

```

```

350 vardef punct.dividedby(expr t) =
351   push_stroke(((1,0)-(1,0)) transformed t,(2,2)-(2,2));
352   set_bosize(0,90);
353
354   push_lcblob(fullcircle scaled (0.65*tsu_punct_size/xxpart t)
355     shifted (0,0.9) transformed t);
356   push_lcblob(fullcircle scaled (0.65*tsu_punct_size/xxpart t)
357     shifted (0,-0.9) transformed t);
358
359   push_pbox_explicit("punct.dividedby",
360     identity shifted (-0.5,-0.5) scaled 2.4 transformed t);
361 enddef;
362
363 vardef punct.equals(expr t) =
364   push_stroke(((1,0.667)-(1,0.667)) transformed t,(2,2)-(2,2));
365   set_bosize(0,90);
366
367   push_stroke(((1,-0.667)-(1,-0.667)) transformed t,(2,2)-(2,2));
368   set_bosize(0,90);
369
370   push_pbox_explicit("punct.equals",
371     identity shifted (-0.5,-0.5) xyscaled (2.4,1.8) transformed t);
372 enddef;

```



```

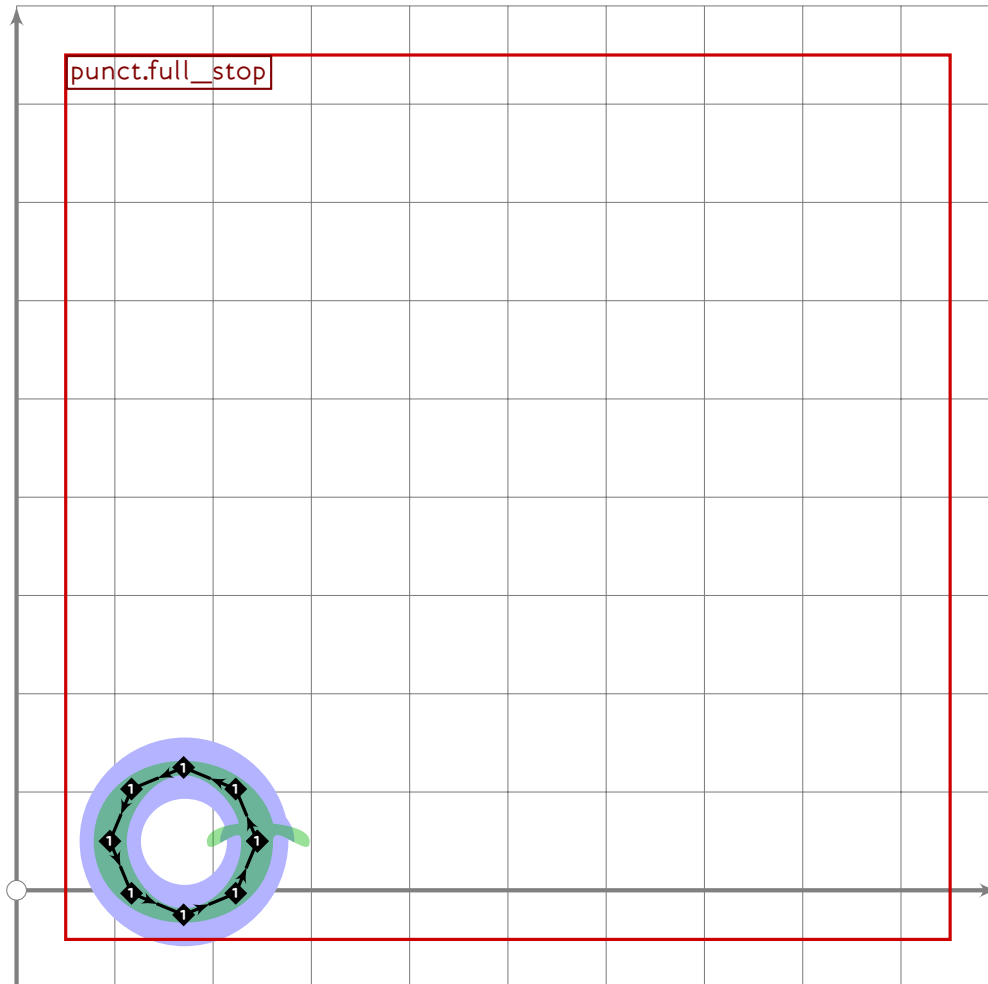
373
374 vardef punct.euro =
375   push_pbox_toexpand("punct.euro");
376
377   push_stroke((subpath (0.5,3.5) of ((1,0)..(0,1)..(-1,0)..(0,-1)..cycle))
378     scaled ((latin_wide_high_r-latin_wide_low_r)/2)
379     shifted (centre_pt+(100,0)),
380     (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
381   set_bosize(0,90);
382
383   push_stroke(((0,-1.25,0.1667)-
384     (((0,0.1667)-(1,0.1667)) intersectionpoint ((0.707,0.707)-(0,-1))))
385     scaled ((latin_wide_high_r-latin_wide_low_r)/2)
386     shifted (centre_pt+(100,0)),
387     (1.6,1.6)-(1.6,1.6));
388   set_bosize(0,90);
389
390   push_stroke(((0,-1.25,-0.1667)-
391     (((0,-0.1667)-(1,-0.1667)) intersectionpoint ((0.707,0.707)-(0,-1))))
392     scaled ((latin_wide_high_r-latin_wide_low_r)/2)
393     shifted (centre_pt+(100,0)),

```

```

394     (1.6,1.6)-(1.6,1.6));
395     set_bosize(0,90);
396     expand_pbox;
397 endif;
398

```



```

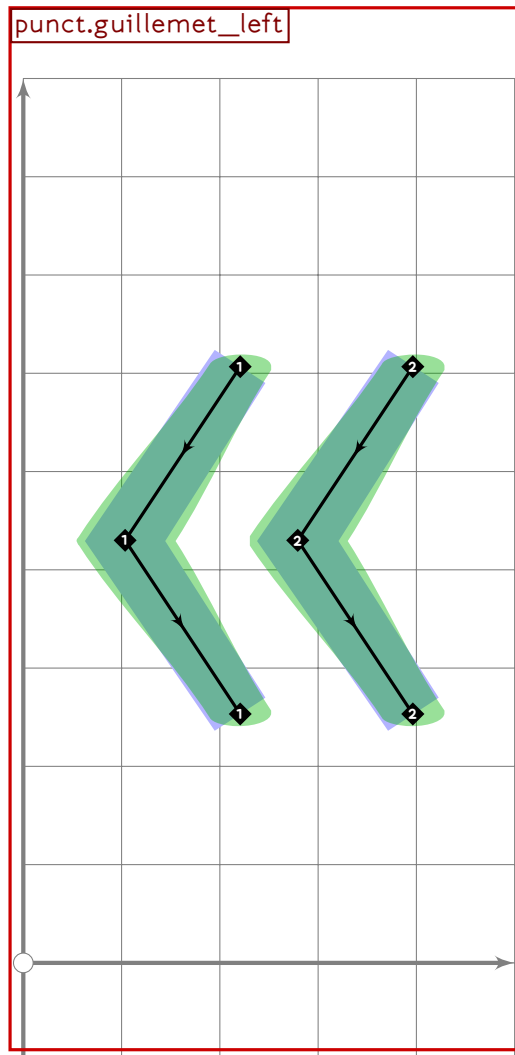
399 % this is *ideographic* full stop, not Latin period
400 vardef punct.full_stop =
401   push_pbox_toexpand("punct.full_stop");
402
403   if tsu_pbrush_size>=tsu_punct_size:
404     push_lcblob(fullcircle
405       xscaled (1.5*tsu_punct_size+tsu_pbrush_size)
406       yscaled (1.5*tsu_punct_size+tsu_pbrush_size*tsu_pbrush_shape)
407       rotated tsu_pbrush_angle
408       shifted (170,50));
409   else:
410     push_stroke(fullcircle scaled (1.5*tsu_punct_size) shifted (170,50),
411       (1,1)-(1,1)-(1,1)-(1,1)-cycle);
412   fi;
413   expand_pbox;
414 endif;

```

U+00AB

tsuku.guillemotleft

```
415
416 vardef punct.greater_than(expr t) =
417   push_stroke(((−1,1)−(1,0)−(−1,1)) transformed t,(2,2)−(2,2)−(2,2));
418   set_bosize(0,90);
419   set_botip(0,1,1);
420
421   push_pbox_explicit("punct.greater_than",
422     identity shifted (−0.5,0.5) scaled 2.4 transformed t);
423 enddef;
```



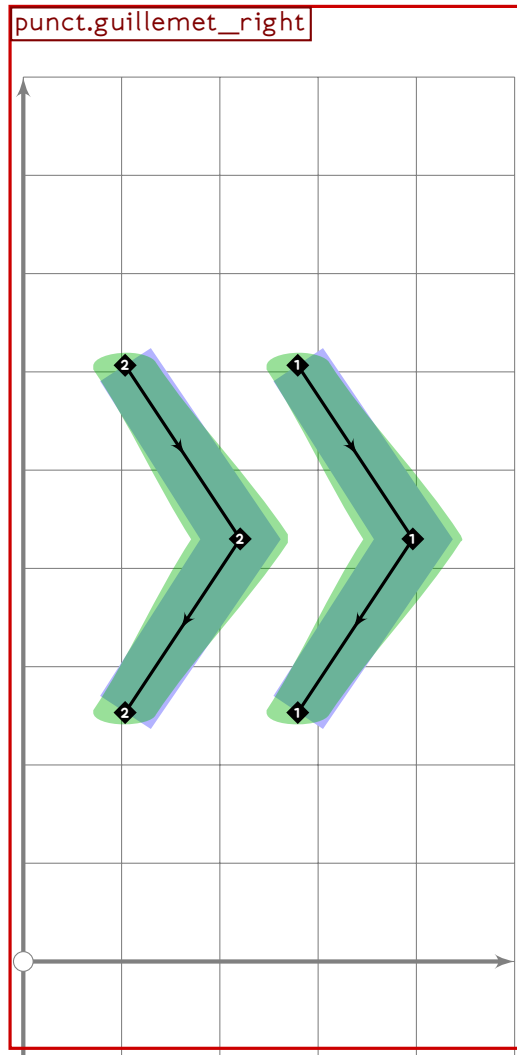
```
424
425 vardef punct.guillemet_left =
426   push_pbox_toexpand("punct.guillemet_left");
427
428   push_stroke(((−0.5,1.5)−(−2.5,0)−(−0.5,1.5))
429     scaled tsu_punct_size shifted centre_pt,
430     (1.5,1.5)−(2,2)−(1.5,1.5));
431   set_bosize(0,90);
432   set_botip(0,1,1);
433
```

PUNC

```

434 push_stroke(((2.5,1.5)-(0.5,0)-(2.5,-1.5))
435     scaled tsu_punct_size shifted centre_pt,
436     (1.5,1.5)-(2,2)-(1.5,1.5));
437 set_bosize(0,90);
438 set_botip(0,1,1);
439 expand_pbox;
440 endif;

```



```

441
442 vardef punct.guillemet_right =
443   push_pbox_toexpand("punct.guillemet_right");
444
445   push_stroke(((0.5,1.5)-(2.5,0)-(0.5,-1.5))
446       scaled tsu_punct_size shifted centre_pt,
447       (1.5,1.5)-(2,2)-(1.5,1.5));
448   set_bosize(0,90);
449   set_botip(0,1,1);
450
451   push_stroke(((2.5,1.5)-(0.5,0)-(2.5,-1.5))
452       scaled tsu_punct_size shifted centre_pt,

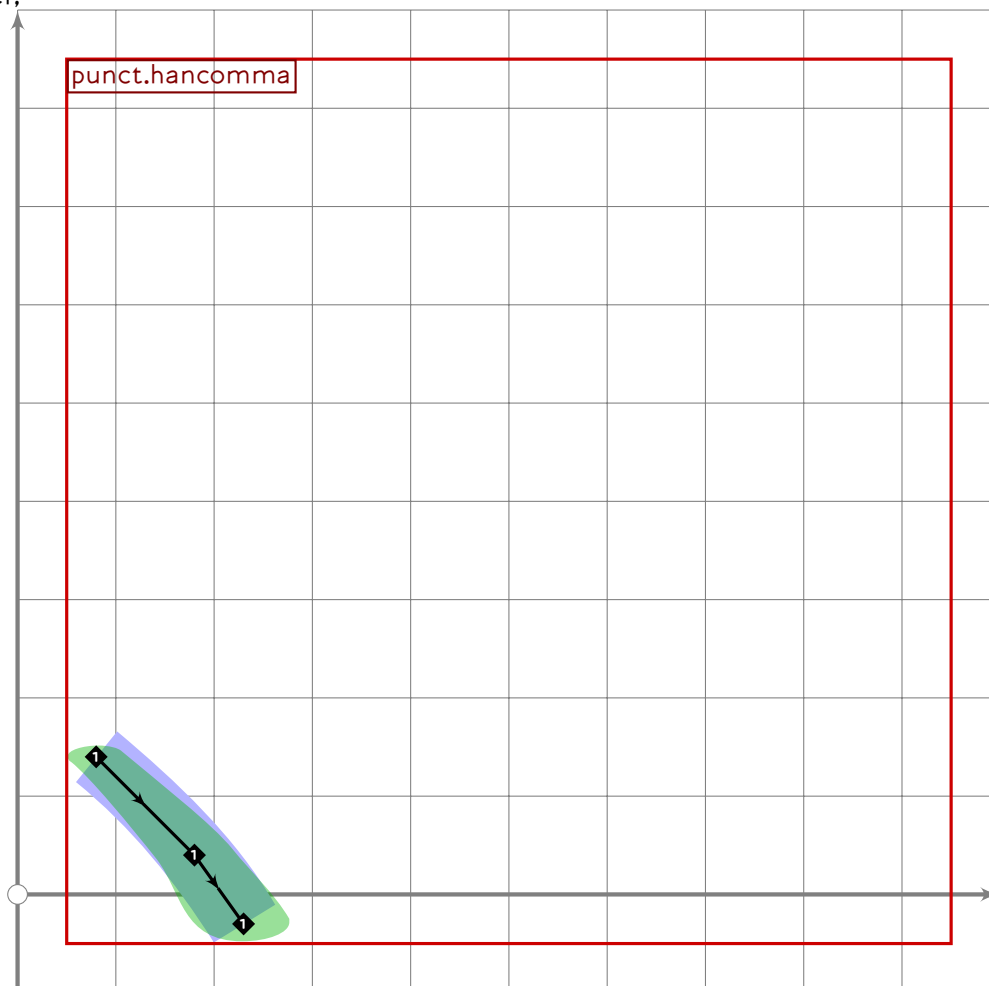
```

U+3001  
tsuku.uni3001

```

453   (1.5,1.5)-(2,2)-(1.5,1.5));
454   set_bosize(0,90);
455   set_botip(0,1,1);
456   expand_pbox;
457 enddef;

```



```

458
459 vardef punct.hancomma =
460   push_pbox_toexpand("punct.hancomma");
461   push_stroke((80,140)..(180,40)..(230,-30),(1.3,1.3)..(1.6,1.6)..(1.8,1.8));
462   expand_pbox;
463 enddef;
464
465 vardef punct.hminus(expr t) =
466   push_stroke(((1,0)-(1,0)) transformed t,(2,2)-(2,2));
467
468   push_pbox_explicit("punct.hminus",
469     identity shifted (-0.5,-0.5) xyscaled (24,0.6) transformed t);
470 enddef;
471
472 vardef punct.less_than(expr t) =
473   push_stroke(((1,1)-(1,0)-(1,1)) transformed t,(2,2)-(2,2)-(2,2));

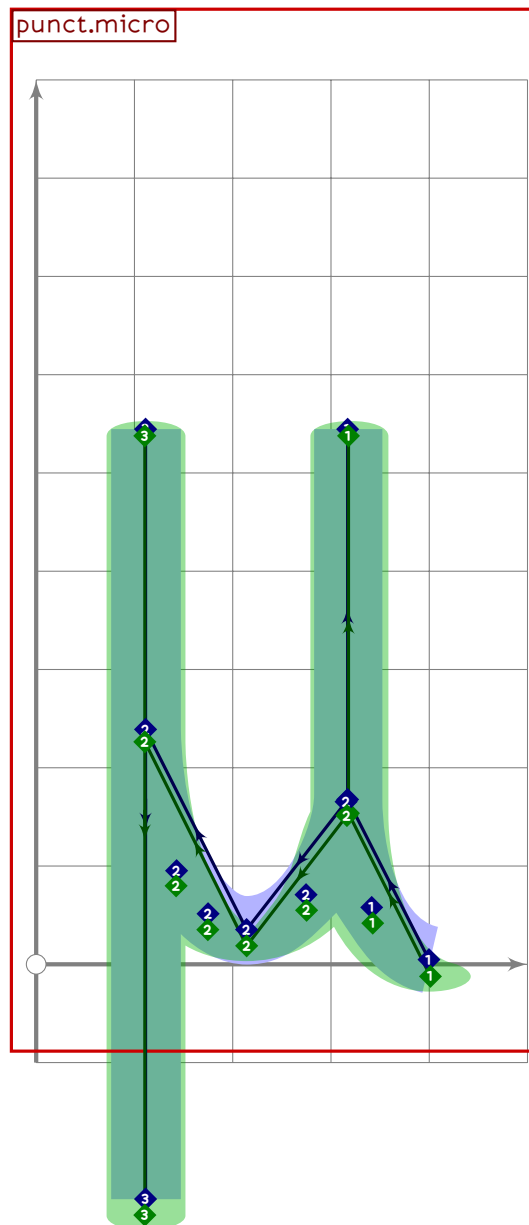
```

**PUNC**

```

474 set_bosize(0,90);
475 set_botip(0,1,1);
476
477 push_pbox_explicit("punct.less_than",
478   identity shifted (-0.5,-0.5) scaled 2.4 transformed t);
479 enddef;
480

```



```

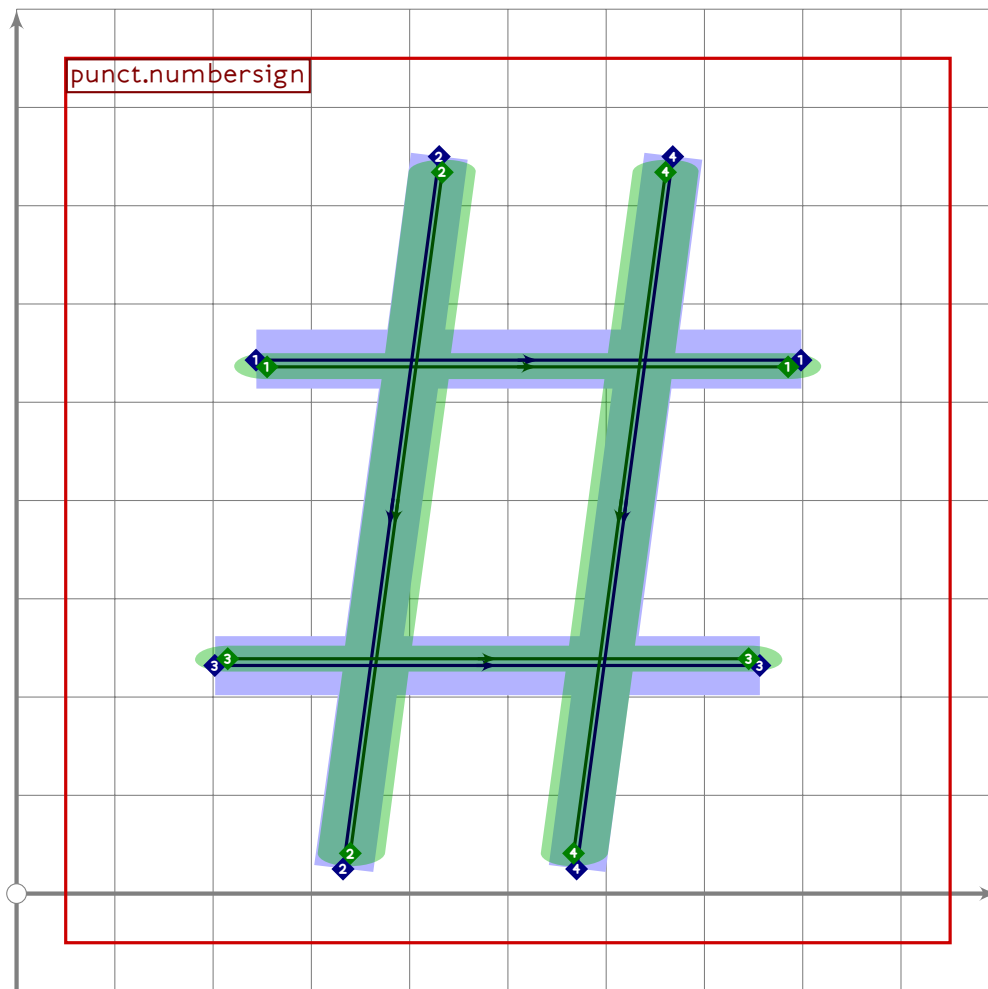
481 % in the future, this will probably become greek.lowmu
482 vardef punct.micro =
483   push_pbox_toexpand("punct.micro");
484
485   x1-x2=y2-y1;
486   (x2+x6)/2=450;
487   (x2-x6)=(y3-y1)*0.75;
488   x3=x2=x4;

```

```

489 x5=0.5[x4,x6];
490 x7=x8=x6;
491
492 y1=(-0.06)[y5,y3];
493 y2=0.26[y5,y3];
494 y3=y7=latin_wide_xheight_v;
495 y4=0.73[y3,y5];
496 y5=latin_wide_low_h;
497 y6=0.60[y3,y5];
498 y8=latin_wide_desc_v;
499
500 push_stroke(z1{dir 173}..{up}z2-z3,(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
501 push_stroke(subpath (0.03,2) of (z4..z5{left}..z6{dir 93}),
502   (1.6,1.6)-(1.6,1.6)-(1.6,1.6));
503 push_stroke(z7-z8,(1.6,1.6)-(1.6,1.6));
504 expand_pbox;
505 enddef;
506
507 vardef punct.otsign(expr t) =
508   push_stroke(((1,0)-(1,0)-(1,1)) transformed t,(2,2)-(2,2)-(2,2));
509   set_bosize(0,90);
510   set_botip(0,1,1);
511
512   push_pbox_explicit("punct.otsign",
513     identity shifted (-0.5,-0.5) scaled 2.4 transformed t);
514 enddef;

```



```

515
516 vardef punct.numbersign =
517   push_pbox_toexpand("punct.numbersign");
518
519   (x1+x2)/2=500;
520   (x2-x1)=0.9*(y2-y1);
521   x3=0.15[x1,x2];
522
523   y1=latin_wide_low_v;
524   y2=y3=latin_wide_high_v;
525
526   transform xf_num;
527   (0,0) transformed xf_num = z1;
528   (3.5,3.5) transformed xf_num = z2;
529   (0,3.5) transformed xf_num = z3;
530
531   push_stroke(((0,2.5)-(3.5,2.5)) transformed xf_num,(1.6,1.6)-(1.6,1.6));
532   set_bosize(0,85);
533   push_stroke(((1,3.5)-(1,0)) transformed xf_num,(1.6,1.6)-(1.6,1.6));
534   set_bosize(0,85);
535   push_stroke(((0,1)-(3.5,1)) transformed xf_num,(1.6,1.6)-(1.6,1.6));

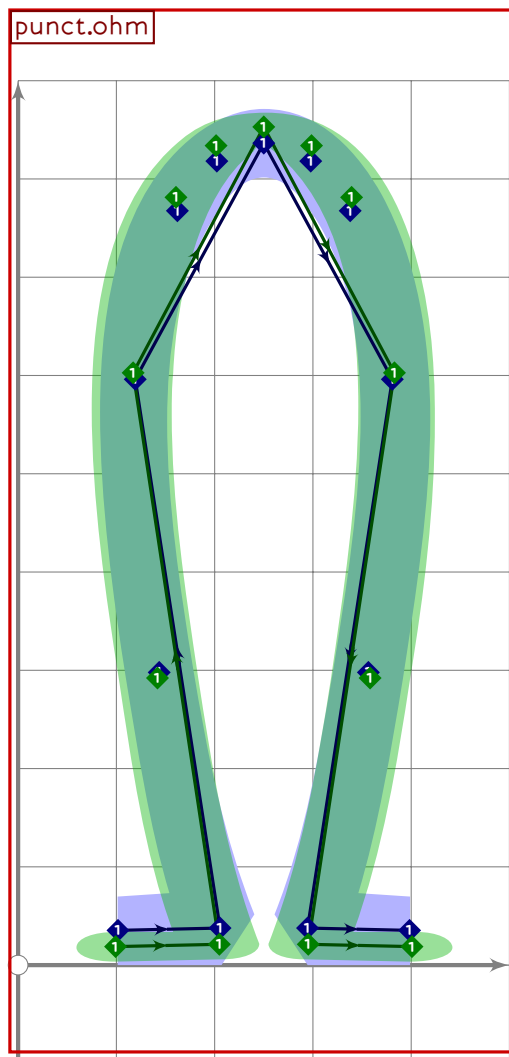
```

U+2126  
tsuku.uni2126

```

536 set_bosize(0,85);
537 push_stroke(((2.5,3.5)-(2.5,0)) transformed xf_num,(1.6,1.6)-(1.6,1.6));
538 set_bosize(0,85);
539 expand_pbox;
540 enddef;
541

```



```

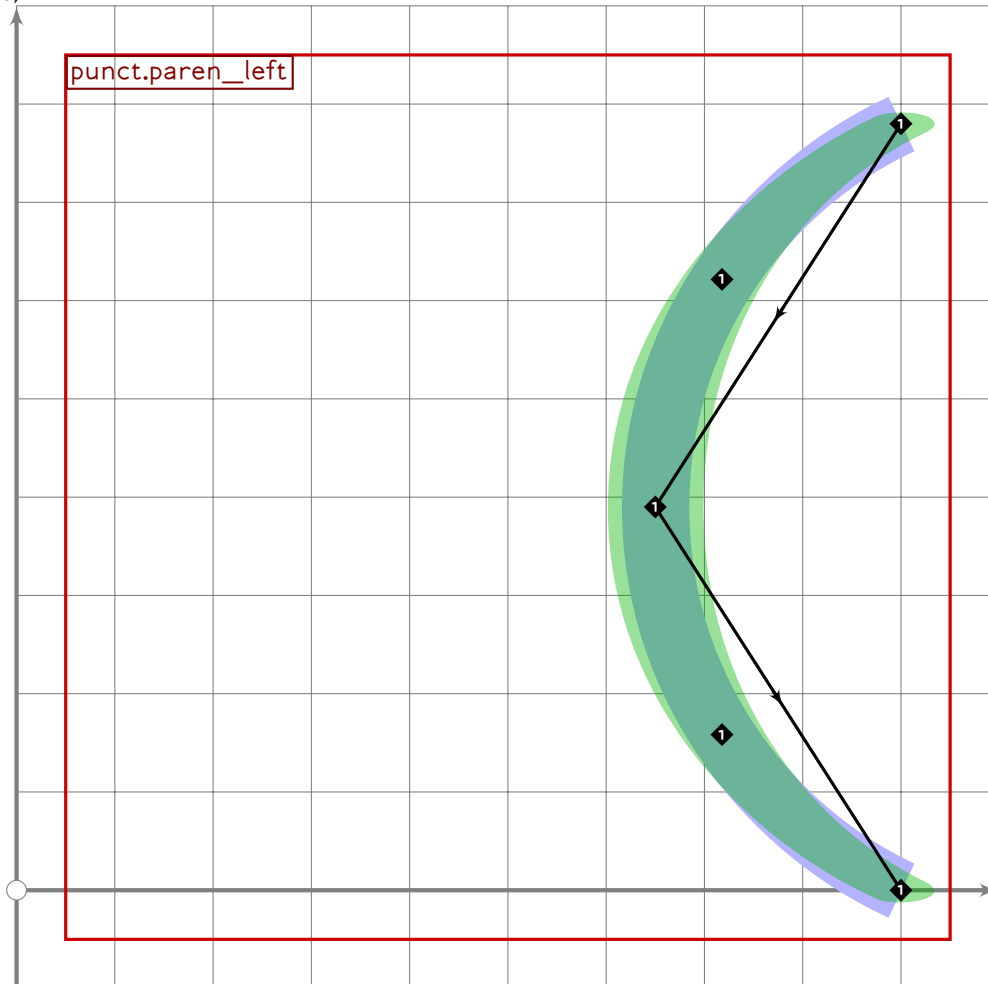
542 % in the future, this will probably become greek.upomega
543 vardef punct.ohm =
544   push_pbox_toexpand("punct.ohm");
545
546   (x5+x3)/2=(x6+x2)/2=(x7+x1)/2=x4=500;
547   x2=0.7[x1,x4];
548   x7-x1=0.76*(y4-y1);
549   x5-x3=0.67*(y4-y1);
550
551   y1=y7=latin_wide_low_h;
552   y2=y6=y1+2;
553   y3=y5=0.7[y1,y4];
554   y4=latin_wide_high_r;

```

```

555
556 push_stroke(z1-z2..tension 1.5 and 1.3..z3..z4..
557       z5..tension 1.3 and 1.5..z6-z7,
558       (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-
559       (1.6,1.6)-(1.6,1.6));
560 set_botip(0,1,0);
561 set_botip(0,5,0);
562 expand_pbox;
563 endif;

```

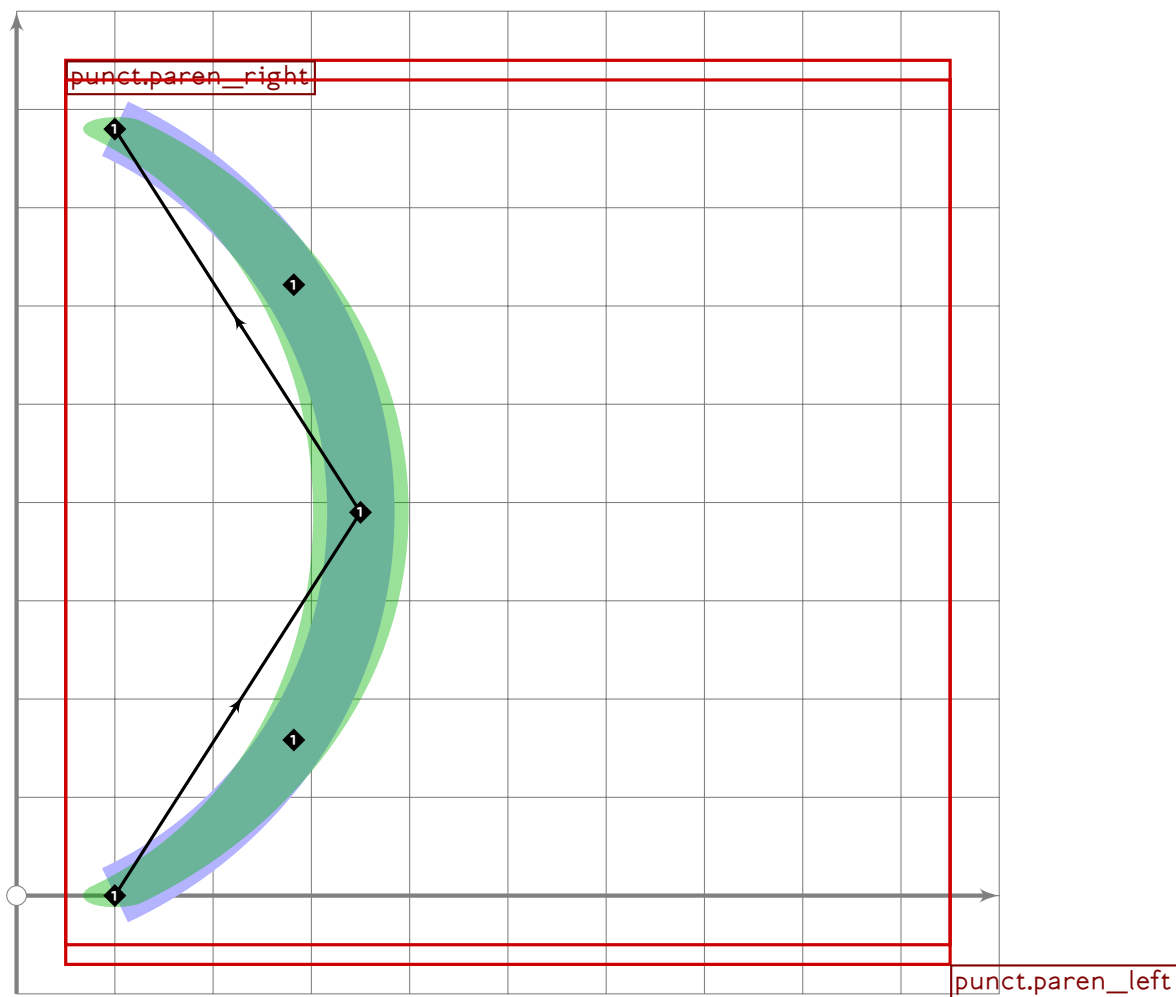


```

564
565 vardef punct.paren_left =
566   push_pbox_toexpand("punct.paren_left");
567   push_stroke((900,780)..(900-2.5*tsu_punct_size,390)..(900,0),
568       (1.5,1.5)-(2,2)-(1.5,1.5));
569   set_bosize(0,90);
570   expand_pbox;
571 endif;

```

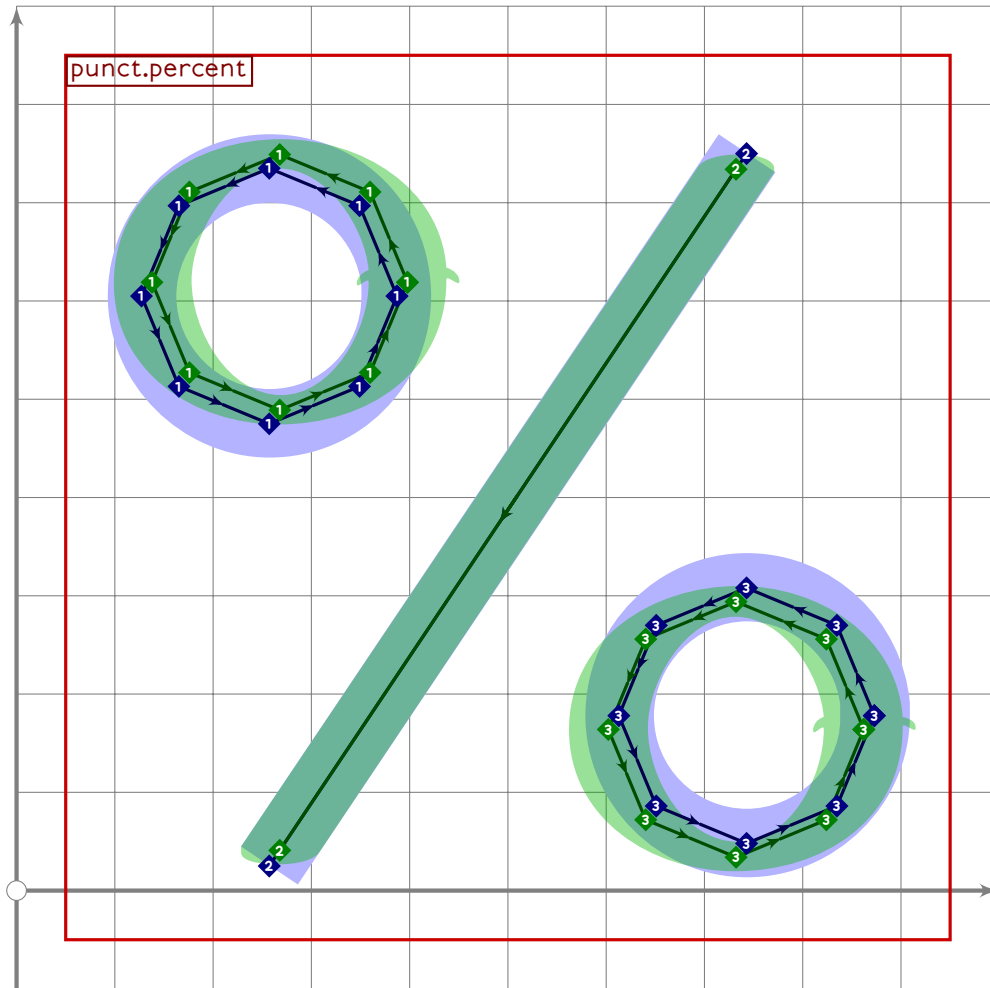
U+FF09  
tsuku.uniFF09



```

572
573 vardef punct.paren_right =
574   push_pbox_toexpand("punct.paren_right");
575   tsu_xform(identity rotatedaround (centre_pt,180))
576   (punct.paren_left);
577   expand_pbox;
578 enddef;

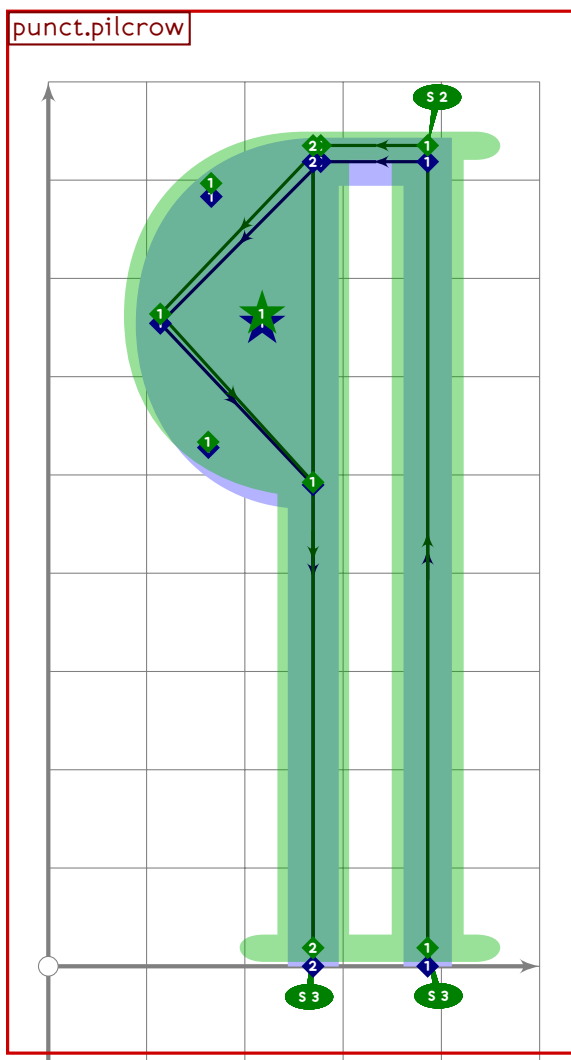
```



```

579
580 vardef punct.percent =
581   push_pbox_toexpand("punct.percent");
582
583   (x1+x2)/2=500;
584   (x1-x2)=0.67(y1-y2);
585
586   y1=latin_wide_high_v;
587   y2=latin_wide_low_v;
588
589   push_stroke(fullcircle scaled 260 shifted (x2,latin_wide_high_r-130),
590     (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-cycle);
591
592   push_stroke((z1-z2)
593     shifted -centre_pt scaled (tsu_punct_size/100) shifted centre_pt,
594     (1.6,1.6)-(1.6,1.6));
595
596   push_stroke(fullcircle scaled 260 shifted (x1,latin_wide_low_r+130),
597     (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-cycle);
598   expand_pbox;
599 enddef;

```



```

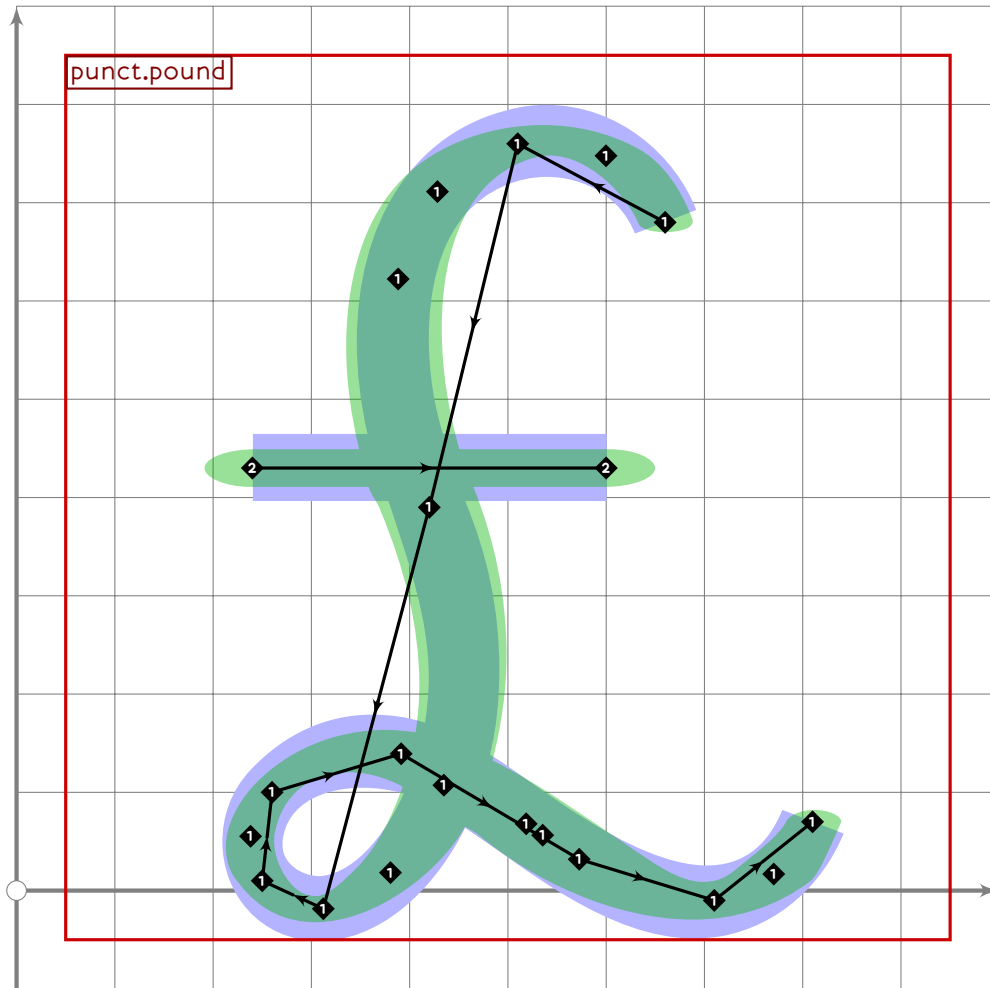
600
601 vardef punct.pilcrow =
602   push_pbox_toexpand("punct.pilcrow");
603
604   x1=x2=x6=710;
605   x3=x5=x1-420*0.4;
606   x4=x1-420;
607
608   y1=latin_wide_low_v;
609   y2=y3=latin_wide_high_h;
610   y4=(y3+y5)/2;
611   y5=y6=vmetric(0.58);
612
613   x7=x8=x9=x1-1.8*tsu_punct_size;
614   y7=y2+50;
615   y8=y4;
616   y9=y1;
617
618   push_stroke(z1-z2-z3{left}..z4.{right}z5-z6,

```

```

619 (2,2)-(2,2)-(2,2)-(2,2)-(2,2)-(2,2));
620 replace_strokeq(0)(subpath (0,xpart (get_strokep(0) intersectiontimes
621 (z8-z9))) of oldq);
622 replace_strokep(0)(subpath (0,xpart (oldp intersectiontimes
623 (z8-z9))) of oldp);
624 set__bysize(0,67);
625 set__botip(0,1,1);
626 set__boserif(0,0,3);
627 set__boserif(0,1,2);
628
629 push_stroke(((z7-z8) intersectionpoint get_strokep(0))-z9,(2,2)-(2,2));
630 set__bysize(0,67);
631 set__boserif(0,1,3);
632
633 if tsu_pbrush_size>=30:
634   push_lcblob((subpath (xpart (get_strokep(-1) intersectiontimes (z7-z8)),
635     infinity) of get_strokep(-1))-cycle);
636 fi;
637 expand__pbox;
638 enddef;
639
640 vardef punct.plus(expr t) =
641   push_stroke(((1,0)-(1,0)) transformed t,(2,2)-(2,2));
642   push_stroke(((0,1)-(0,-1)) transformed t,(2,2)-(2,2));
643   push_pbox_explicit("punct.plus",
644     identity shifted (-0.5,-0.5) scaled 2.4 transformed t);
645 enddef;
646
647 vardef punct.plusminus(expr t) =
648   push_stroke(((1,0.25)-(1,0.25)) transformed t,(2,2)-(2,2));
649   push_stroke(((0,1.25)-(0,-0.75)) transformed t,(2,2)-(2,2));
650   push_stroke((-1,-1.25)-(1,-1.25)) transformed t,(2,2)-(2,2));
651
652   push_pbox_explicit("punct.plusminus",
653     identity shifted (-0.5,-0.5) xyscaled (2.4,3.2) transformed t);
654 enddef;

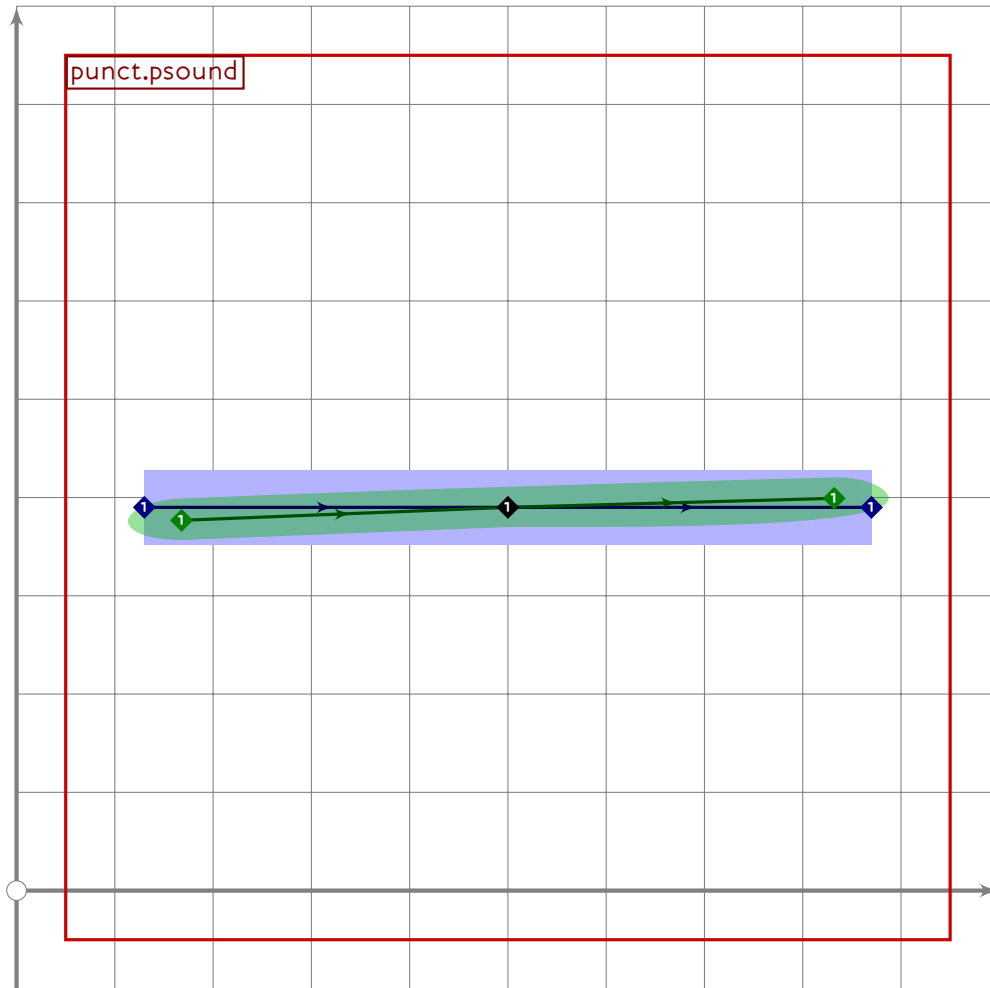
```



```

655
656 vardef punct.pound =
657   push_pbox__toexpand("punct.pound");
658
659   push_stroke((660,680)..(510,760)..(420,390)..tension 1.1..(250,10)..
660     (260,100)..(710,-10)..(810,70),
661     (1.3,1.3)-(1.7,1.7)-(1.9,1.9)-(1.4,1.4)-(1.2,1.2)-
662     (1.1,1.2)-(2,2)-(2.1,2.1)-(2,2)-(1.3,1.3));
663   replace_strokep(0)(insert_nodes(oldp)(2.8,4.3,4.7));
664
665   push_stroke((240,430)-(600,430),(2,2)-(2,2));
666   set_bosize(0,90);
667   expand_pbox;
668 enddef;

```



```

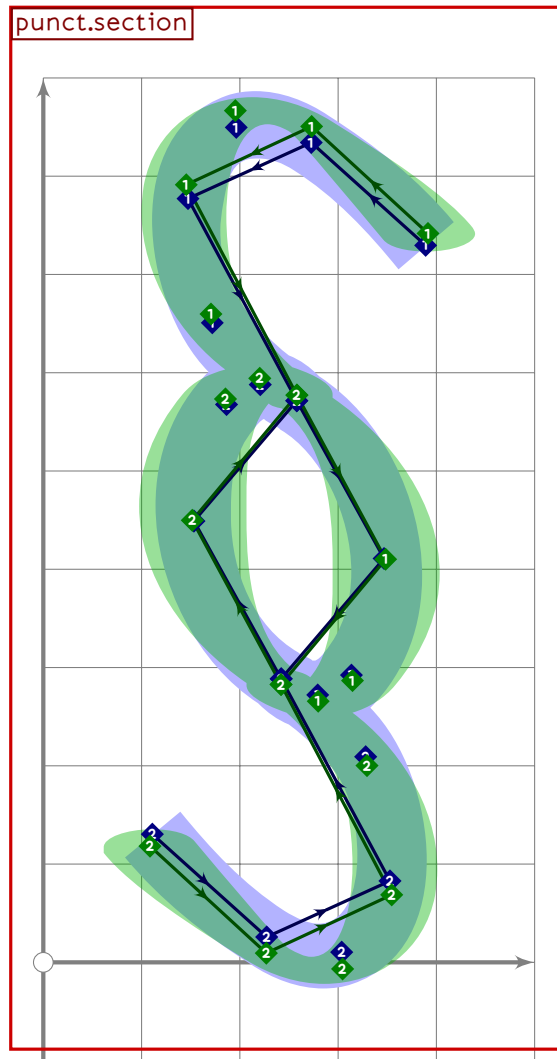
669
670 vardef punct.psound =
671   push_pbox_toexpand("punct.psound");
672   push_stroke((130,390-15*mincho)..(500,390)..(870,390+10*mincho),
673     (0,7,3,3)-(2,2)-(0,7,3,3));
674   expand_pbox;
675 enddef;
676
677 vardef punct.make_period(expr cpos) =
678   push_stroke(fullcircle scaled (tsu_punct_size*1.15) shifted cpos,
679     (2,2)-(2,2)-(2,2)-(2,2)-cycle);
680
681   if tsu_pbrush_size>=30:
682     set_bosize(0,40);
683     push_lcblob(get_strokep(0));
684   else:
685     set_bosize(0,80);
686   fi;
687
688   push_pbox_explicit("punct.make_period",
689     identity shifted (-0.5,-0.5) scaled (tsu_punct_size*1.5) shifted cpos);

```

U+00A7

tsuku.section

690 endif;



691

692 vardef punct.section =

693 push\_pbox\_toexpand("punct.section");

694

695 (x1+x3)/2=x2=x4=x6=500;

696 x5=0.8[x2,x1];

697 2\*(x5-x2)=0.45\*(latin\_wide\_high\_r-latin\_wide\_low\_r);

698

699 y1=y3=0.8[ypart centre\_pt,latin\_wide\_high\_r];

700 y2=latin\_wide\_high\_r;

701 y4=0.35[ypart centre\_pt,latin\_wide\_high\_r];

702 y5=ypart centre\_pt;

703 y4-y5=y5-y6;

704

705 push\_stroke((z1..z2..z3..z4..z5..z6) rotatedaround (centre\_pt,-6),

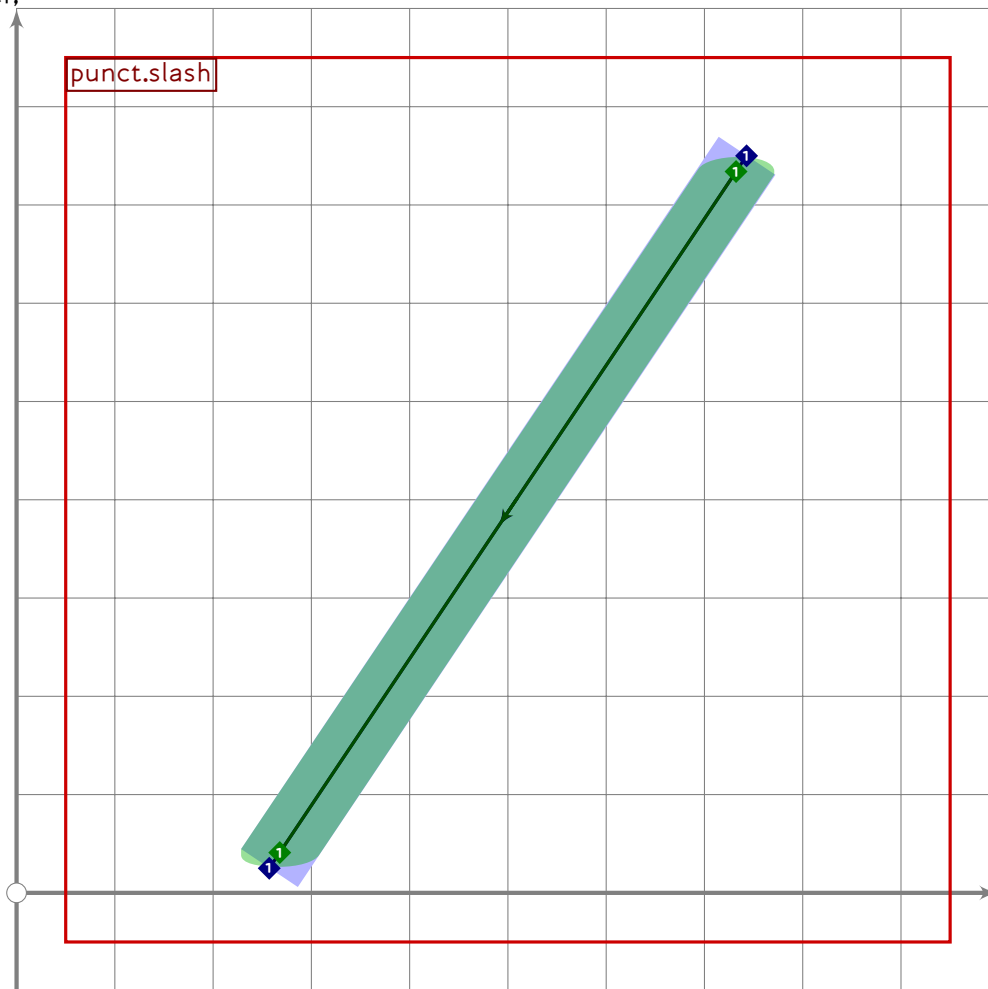
706 (1.8,1.8)-(1.2,1.2)-(1.7,1.7)-(1.3,1.3)-(2,2)-(1.5,1.5));

707

708 push\_stroke(get\_stroke(0) rotatedaround (centre\_pt,180),get\_stroke(0));

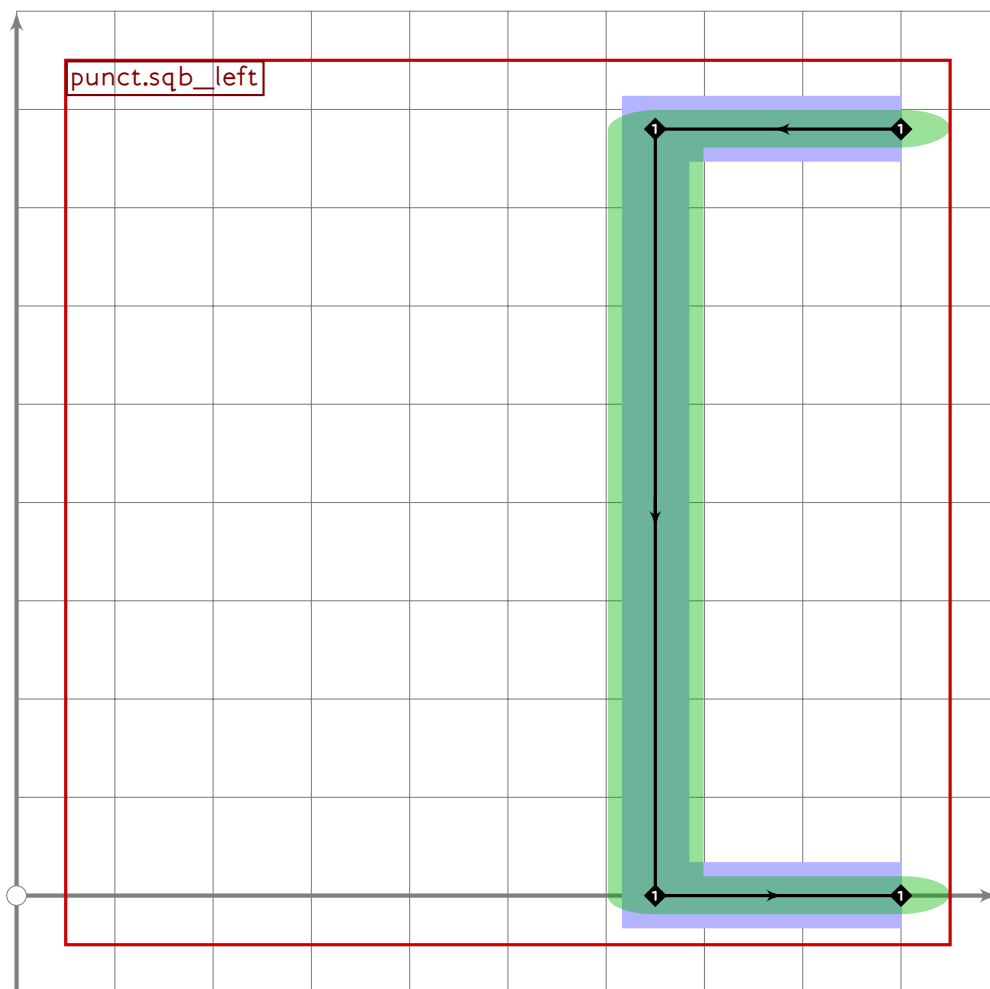
PUNC

```
709 expand_pbox;
710 enddef;
```



```
711
712 vardef punct.slash =
713   push_pbox_toexpand("punct.slash");
714   (x1+x2)/2=500;
715   (x1-x2)=0.67(y1-y2);
716
717   y1=latin_wide_high_v;
718   y2=latin_wide_low_v;
719
720   push_stroke((z1-z2)
721     shifted -centre_pt scaled (tsu_punct_size/100) shifted centre_pt,
722     (1.6,1.6)-(1.6,1.6));
723   expand_pbox;
724 enddef;
```

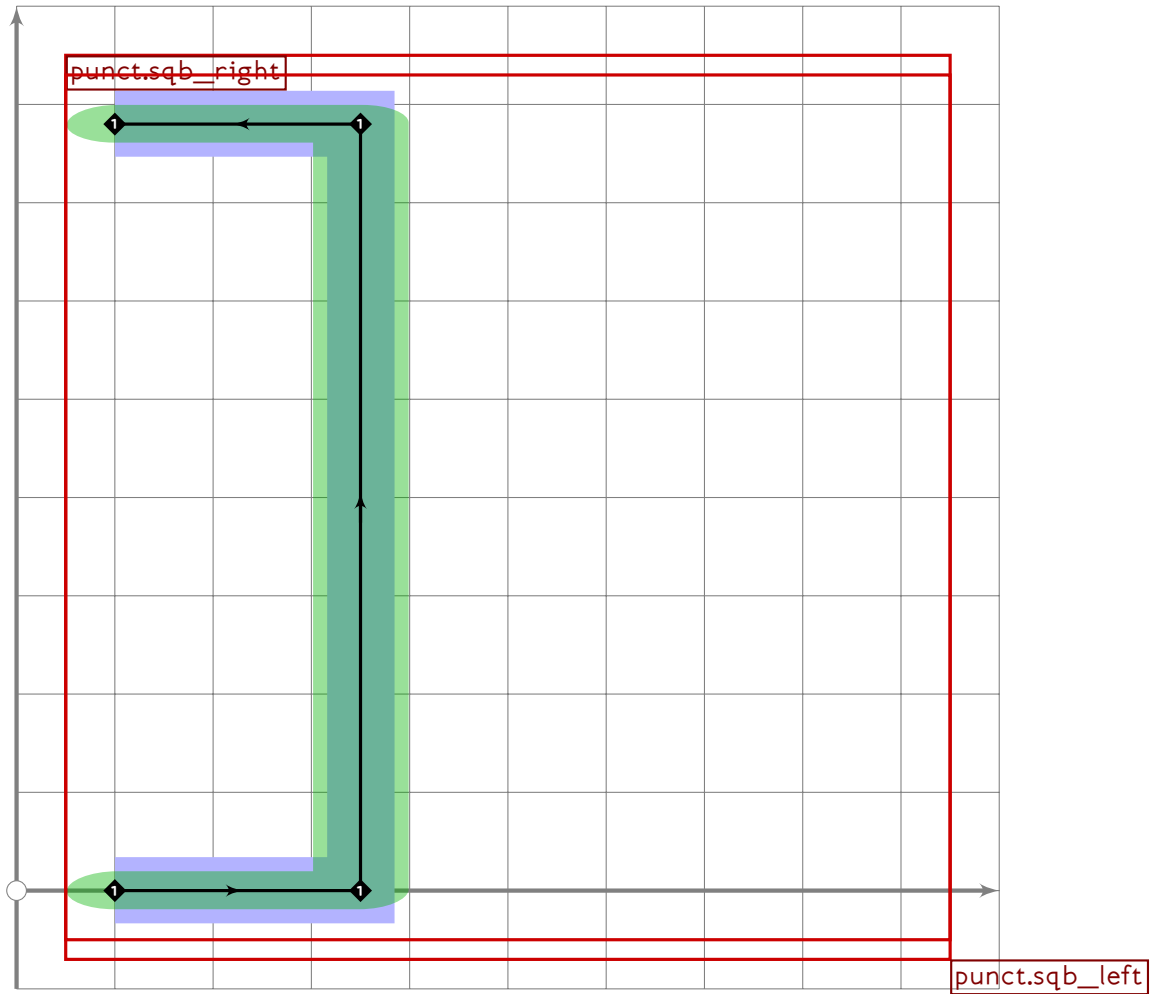
U+FF3B  
tsuku.uniFF3B



```

725
726 vardef punct.sqb_left =
727   push_pbox_toexpand("punct.sqb_left");
728   push_stroke((900,780)-
729     (900-2.5*tsu_punct_size,780)-
730     (900-2.5*tsu_punct_size,0)-
731     (900,0),
732     (2,2)-(2,2)-(2,2)-(2,2));
733   set_bosize(0,90);
734   set_botip(0,1,1);
735   set_botip(0,2,1);
736   expand_pbox;
737 enddef;

```

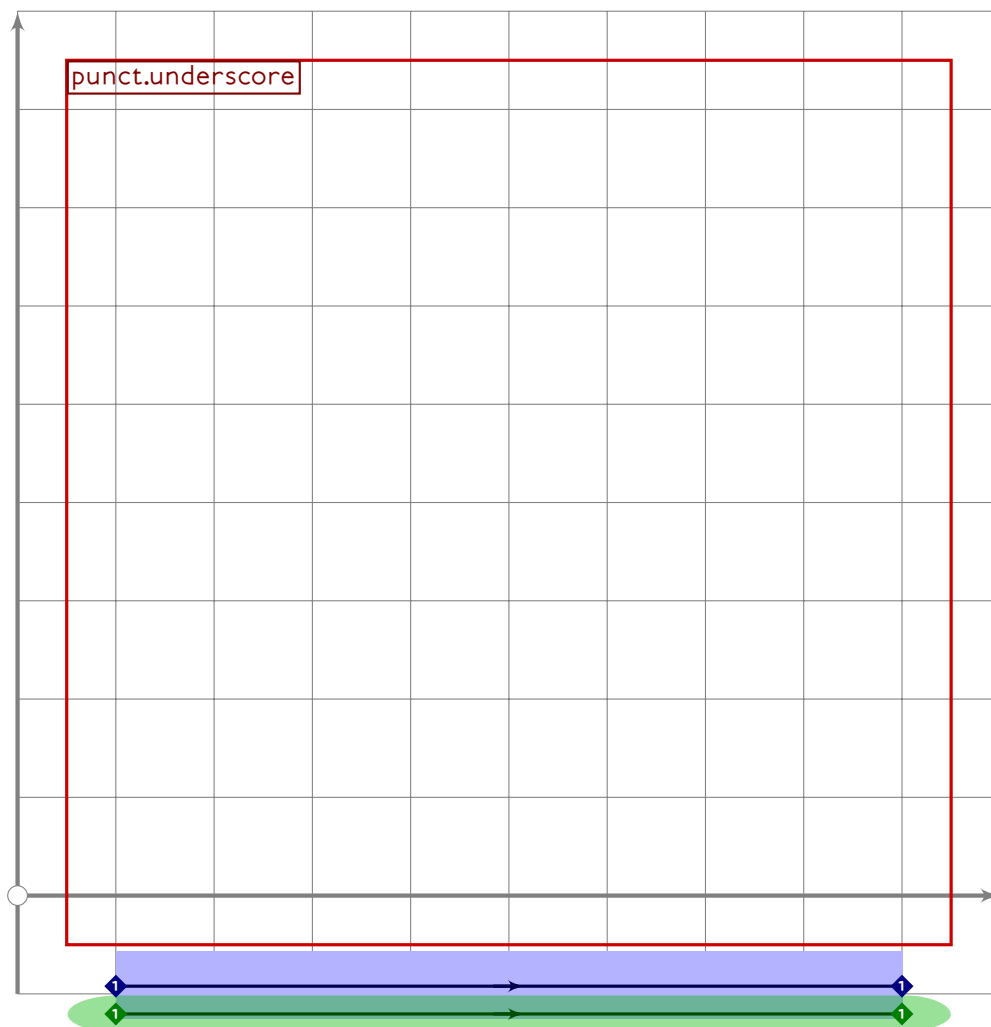


```

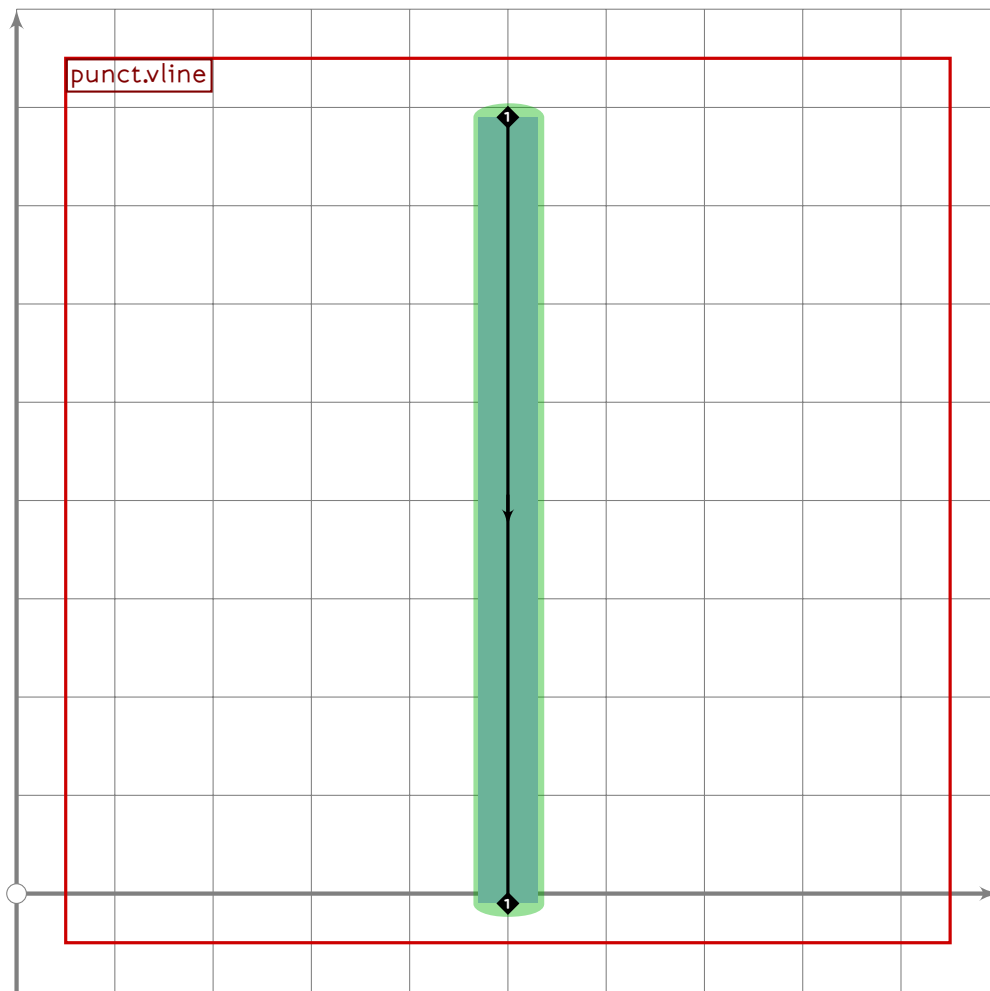
738
739 vardef punct.sqb_right =
740   push_pbox_toexpand("punct.sqb_right");
741   tsu_xform(identity rotatedaround (centre_pt,180))
742   (punct.sqb_left);
743   expand_pbox;
744 enddef;
745
746 vardef punct.times(expr t) =
747   push_stroke((( -1,1)-(1,1)) transformed t,(2,2)-(2,2));
748   set_bosize(0,90);
749   push_stroke((( -1,1)-(1,-1)) transformed t,(2,2)-(2,2));
750   set_bosize(0,90);
751
752   push_pbox_explicit("punct.times",
753     identity shifted (-0.5,-0.5) scaled 2.4 transformed t);
754 enddef;

```

U+FF3F  
tsuku.uniFF3F



```
755
756 vardef punct.underscore =
757   push_pbox_toexpand("punct.underscore");
758   push_stroke((100,0.3[latin_wide_desc_h,latin_wide_low_h])–
759     (900,0.3[latin_wide_desc_h,latin_wide_low_h]),
760     (2,2)–(2,2));
761   set_bosize(0,90);
762   expand_pbox;
763 enddef;
```

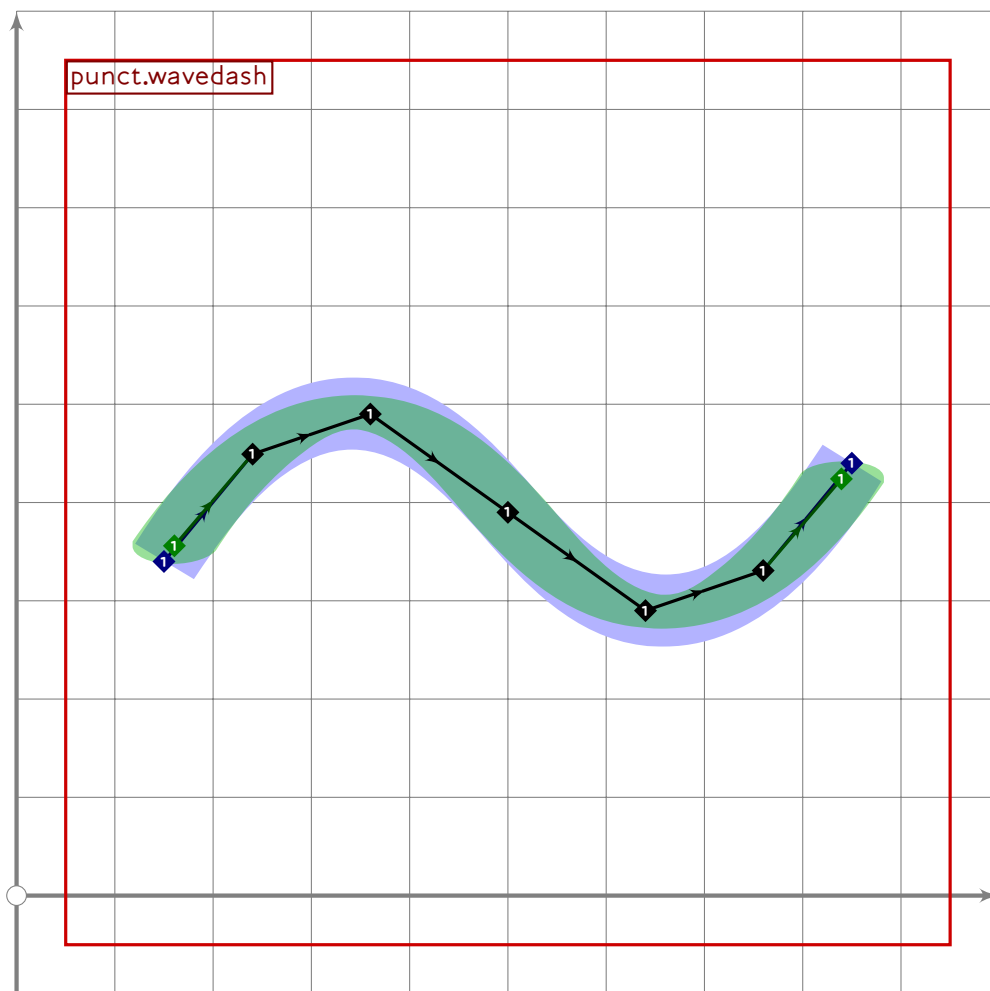


```

764
765 vardef punct.vline =
766   push_pbox__toexpand("punct.vline");
767   push_stroke((500,690+tsu_punct_size)-(500,90-tsu_punct_size),
768     (1.6,1.6)-(1.6,1.6));
769   set_bosize(0,90);
770   expand_pbox;
771 enddef;

```

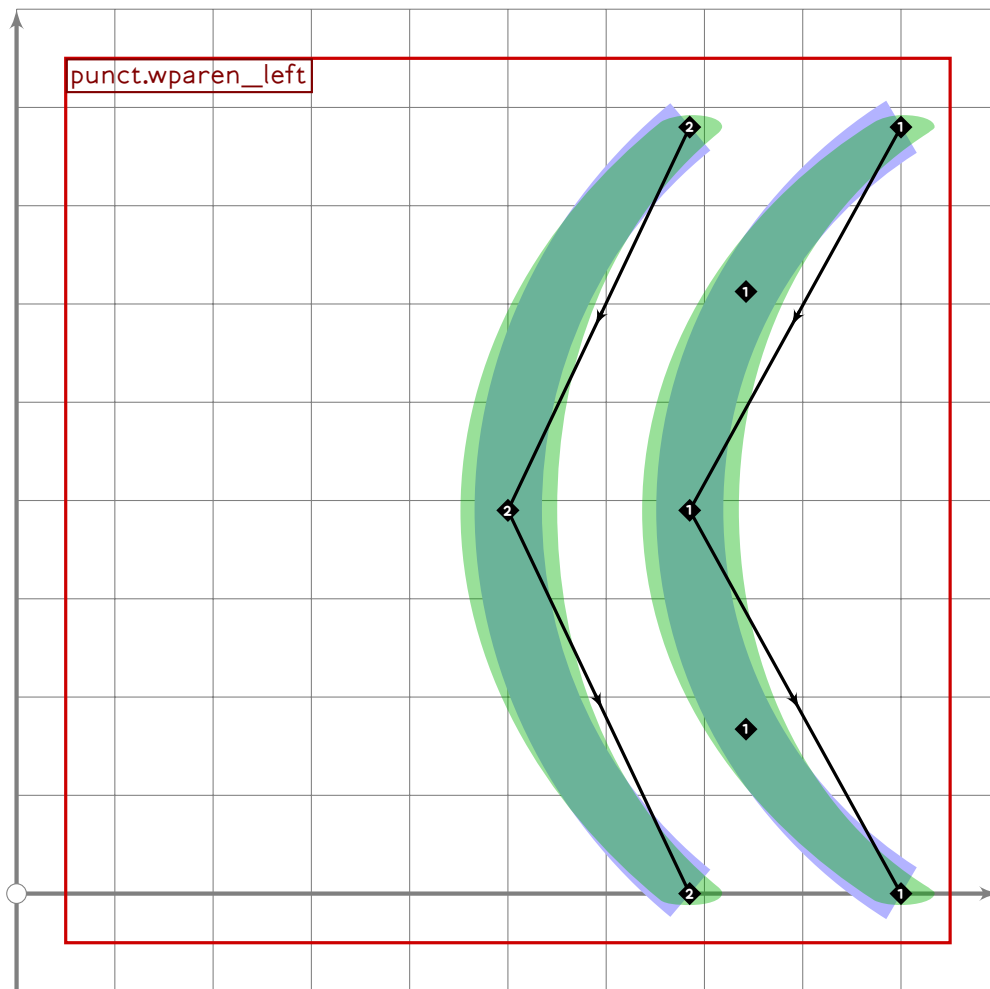
U+FF5E  
tsuku.uniFF5E



```

772
773 vardef punct.wavedash =
774   push_pbox__toexpand("punct.wavedash");
775   push_stroke((( (-3.5,-0.5){curl 0}{-14,1}..(0,0)..(14,-1)..
776     {curl 0}{3.5,0.5})) scaled tsu_punct_size shifted centre_pt,
777     (0.7,2.7)-(1.7,1.7)-(1.7,1.7)-(1.7,1.7)-(1.7,1.7)-
778     (1.7,1.7)-(0.7,2.7));
779   replace_strokep(0)(insert_nodes(oldp)(0.5,3.5));
780   expand_pbox;
781 enddef;

```

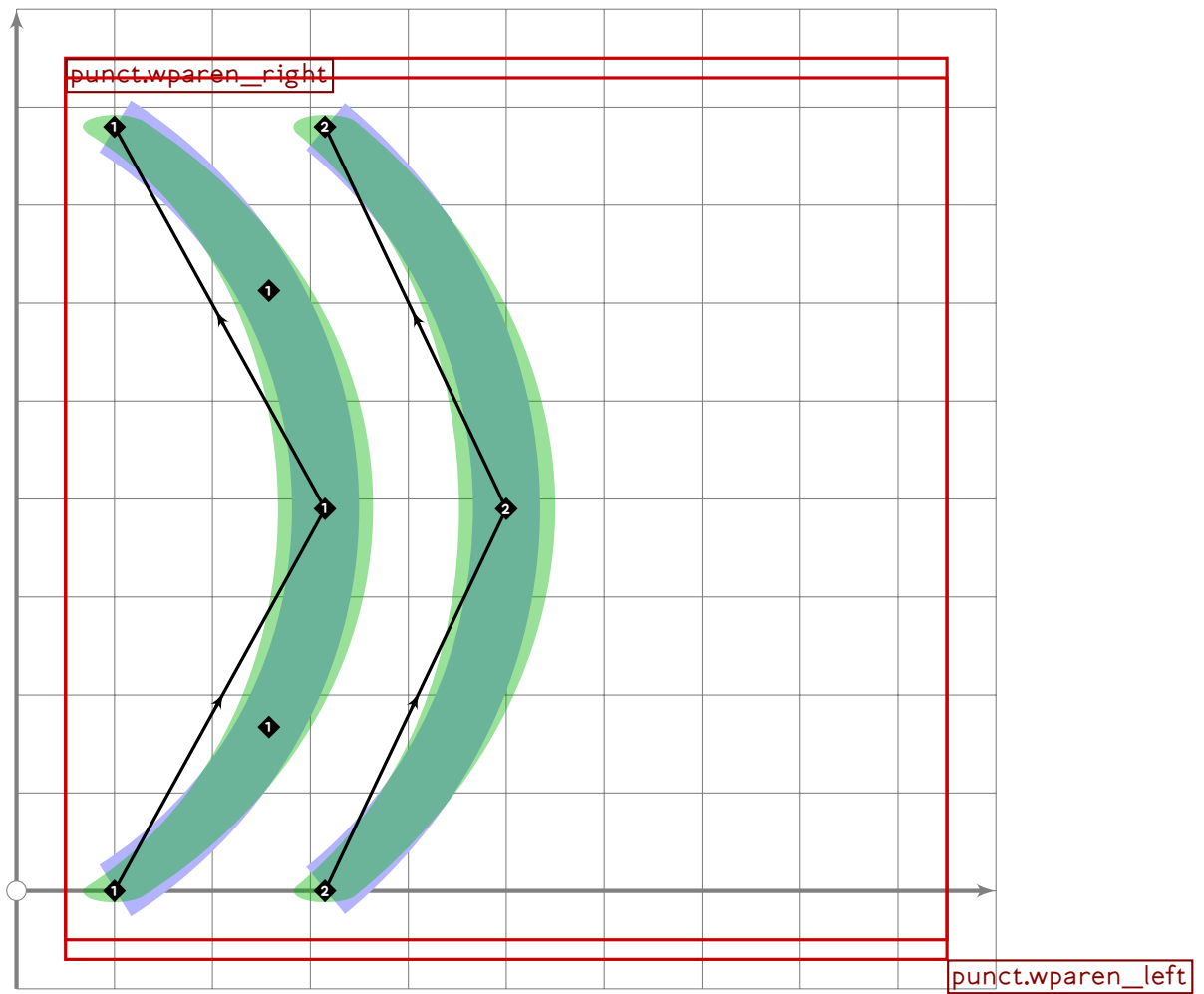


```

782
783 vardef punct.wparen_left =
784   push_pbox_toexpand("punct.wparen_left");
785
786   push_stroke((900,780)..(900-2.15*tsu_punct_size,390)..(900,0),
787     (1.5,1.5)-(2,2)-(1.5,1.5));
788   set_bosize(0,90);
789
790   push_stroke((900-2.15*tsu_punct_size,780)..
791     (900-4*tsu_punct_size,390)..
792     (900-2.15*tsu_punct_size,0),
793     (1.5,1.5)-(2,2)-(1.5,1.5));
794   set_bosize(0,90);
795   expand_pbox;
796 enddef;

```

U+FF60  
tsuku.uniFF60



```

797
798 vardef punct.wparen_right =
799   push_pbox_toexpand("punct.wparen_right");
800   tsu_xform(identity rotatedaround (centre_pt,180))
801   (punct.wparen_left);
802   expand_pbox;
803 enddef;

```

## serif.mp

```

1 %
2 % Serifs for Tsukurimashou
3 % Copyright (C) 2011, 2012 Matthew Skala
4 %
5-29 [Standard copyright notice]
30
31 inclusion_lock(serif);
32
33 

---


34
35 % figure out size of brush
36 (sbrush_width,sbrush_height)=urcorner (
37   fullcircle yscaled tsu_brush_shape
38   rotated tsu_brush_angle
39 );
40 if sbrush_width>sbrush_height:
41   sbrush_long:=sbrush_width;
42   sbrush_short:=sbrush_height;
43 else:
44   sbrush_short:=sbrush_width;
45   sbrush_long:=sbrush_height;
46 fi;
47
48 

---


49
50 vardef tsu_serif.latin.lrcore(expr bst,plp,dlp,l,bts,bos) =
51   serif:=serif xscaled sbrush_long yscaled sbrush_short;
52   serif:=serif xscaled ((1+3*xxpart tsu_rescaling_xf)/4)
53     yscaled bos scaled bts;
54   glstk[ngls]:=regenerate(serif shifted plp);
55   ngls:=ngls+1;
56 enddef;
57
58 vardef tsu_serif.latin.left(expr bst,plp,dlp,l,bts,bos) =
59   begingroup
60     save serif;
61     path serif;
62     if sharp_corners:
63       serif:=(-serif_size,1)-(-serif_size,-1)-
64         (0,-1)-(0,1)-cycle;
65     else:
66       serif:=(-serif_size,1){left}..{right}(-serif_size,-1)-
67         (0,-1)-(0,1)-cycle;
68     fi;
69     tsu_serif.latin.lrcore(bst,plp,dlp,l,bts,bos);
70   endgroup;

```

```

71 enddef;
72
73 vardef tsu_serif.latin.right(expr bst,plp,dlp,l,bts,bos) =
74   begingroup
75     save serif;
76     path serif;
77     if sharp_corners:
78       serif:=(0,1)-(0,-1)-
79         (serif_size,-1)-(serif_size,1)-cycle;
80     else:
81       serif:=(0,1)-(0,-1)-
82         (serif_size,-1){right}..{left}(serif_size,1)-cycle;
83     fi;
84     tsu_serif.latin.lrcore(bst,plp,dlp,l,bts,bos);
85   endgroup;
86 enddef;
87
88 vardef tsu_serif.latin.leftright(expr bst,plp,dlp,l,bts,bos) =
89   begingroup
90     save serif,xoffs;
91     path serif;
92     if sharp_corners:
93       serif:=(-serif_size,1)-(-serif_size,-1)-
94         (serif_size,-1)-(serif_size,1)-cycle;
95     else:
96       serif:=(-serif_size,1){left}..{right}(-serif_size,-1)-
97         (serif_size,-1){right}..{left}(serif_size,1)-cycle; fi;
98     numeric xoffs;
99     xoffs=xpart (dlp/abs(dlp));
100    tsu_serif.latin.lrcore(bst,
101      plp+if l=0: right else: left fi*50*sbrush_long*xoffs,
102      dlp,l,bts,bos);
103  endgroup;
104 enddef;
105
106
107
108 vardef tsu_serif.mincho.corner(expr bst,plp,dlp,l,bts,bos) =
109   begingroup
110     save serif;
111     path serif;
112     serif:=(-1,-0.3)..(0.25,-1.3)..(1,-1.2)..tension 1.2..
113       (1,0.6)..(-0.25,1.3)..(-1,1.2)..tension 1.2..cycle;
114     serif:=serif yscaled sqrt(tsu_brush_shape) rotated tsu_brush_angle
115       scaled (bts*0.43*mincho_blob_size);
116     glstk[ngls]:=regenerate(serif shifted plp);
117     ngls:=ngls+1;
118   endgroup;

```

```

119 enddef;
120
121 vardef tsu_serif.mincho.ulpoint(expr bst,plp,dlp,l,bts,bos) =
122   begingroup
123     save serif;
124     path serif;
125     serif=(-1.5,2.7)..tension 2..(-1,0)..(-0.4,-0.3)..tension 1.5..
126       (0.707,0)..(0.2,0.6)..(-0.1,1.1)..tension 2..(-1.2,2.9)..cycle;
127     serif:=serif yscaled sqrt(tsu_brush_shape) rotated tsu_brush_angle
128       scaled (bts*0.5*mincho_blob_size);
129     glstk[ngls]:=regenerate(serif shifted
130       (plp+(-1,0)*0.25*bts*(xpart dlp/abs(dlp))));
131     ngls:=ngls+1;
132   endgroup;
133 enddef;
134
135 vardef tsu_serif.mincho.triangle(expr bst,plp,dlp,l,bts,bos) =
136   begingroup
137     save serif;
138     path serif;
139     serif=(-1.2,0)..(0,-0.8)..(1.2,0.4)..tension 2..(0.2,1.3)..
140       (-0.2,1.3)..tension 2..cycle;
141     serif:=serif yscaled sqrt(tsu_brush_shape) rotated tsu_brush_angle
142       scaled (bts*0.5*mincho_blob_size);
143     glstk[ngls]:=regenerate(serif shifted (plp+0.25*bts*dlp/abs(dlp)));
144     ngls:=ngls+1;
145   endgroup;
146 enddef;
147
148 vardef tsu_serif.mincho.llpoint(expr bst,plp,dlp,l,bts,bos) =
149   begingroup
150     save serif;
151     path serif;
152     serif=(-2.1,-1.9)..(-1.8,-2.1)..tension 2..(-0.1,-1.2)..(0.2,-1.1)..
153       (0.707,0.707)..(-1,-0.3)..tension 2..cycle;
154     serif:=serif yscaled sqrt(tsu_brush_shape) rotated tsu_brush_angle
155       scaled (bts*0.5*mincho_blob_size);
156     glstk[ngls]:=regenerate(serif shifted plp);
157     ngls:=ngls+1;
158   endgroup;
159 enddef;
160
161 vardef tsu_serif.mincho.lpoint(expr bst,plp,dlp,l,bts,bos) =
162   begingroup
163     save serif;
164     path serif;
165     serif=(-1.5,1.7)..tension 2..(-1,0)..(-0.4,-0.3)..tension 1.8..
166       (0.707,0.2)..(0.2,0.8)..(-0.4,1.1)..tension 2..(-1.2,2.0)..cycle;

```

```

167   serif:=reverse serif reflectedabout ((-1,1),(1,-1))
168   xyscaled (0.7,0.9) shifted (0.3,-0.3);
169   serif:=serif yscaled sqrt(tsu_brush_shape) rotated tsu_brush_angle
170   scaled (bts*mincho_blob_size);
171   glstk[ngls]:=regenerate(serif shifted plp);
172   ngls:=ngls+1;
173 endgroup;
174 enddef;
175
176 vardef tsu_serif.mincho.ktriangle(expr bst,plp,dlp,l,bts,bos) =
177   begingroup;
178   save serif,x,y,t,q;
179   path serif;
180   transform t;
181   (0,0) transformed t=(0,0);
182   right transformed t=(dlp/abs(dlp));
183   z1=(dlp/abs(dlp)) rotated 90;
184   if y1<-x1: x1:=-x1; y1:=-y1; fi;
185   up transformed t=z1;
186   q1:=2;
187   q2:=q1++0.5;
188   q3:=0.5*q2/q1;
189   q5:=0.5+--q3;
190   z2=(0,q1);
191   z3=(-q3,q5);
192   z4=(q3,q5);
193   serif:=((0.1[z2,z4]){z2-z4}..{z3-z2}(0.1[z2,z3])--
194     (0.1[z3,z2]){z3-z2}..{z4-z3}(0.1[z3,z4])--
195     (0.1[z4,z3]){z4-z3}..{z2-z4}(0.1[z4,z2])--cycle)
196     shifted (0,-0.19)
197     transformed t yscaled tsu_brush_shape
198     rotated tsu_brush_angle
199     scaled (bts*bos*0.94*mincho_blob_size) shifted plp;
200   if (xxpart t)*(yypart t)<0: serif:=reverse serif; fi;
201   glstk[ngls]:=regenerate(serif);
202   ngls:=ngls+1;
203 endgroup;
204 enddef;
205
206 vardef tsu_serif.mincho.khellipse(expr bst,plp,dlp,l,bts,bos) =
207   begingroup;
208   save serif,x,y,t,q;
209   path serif;
210   transform t;
211   (0,0) transformed t=(0,0);
212   right transformed t=(dlp/abs(dlp));
213   z1=(dlp/abs(dlp)) rotated 90;
214   up transformed t=z1;

```

```

215   t:=t rotated -tsu_brush_angle yscaled tsu_brush_shape
216     rotated tsu_brush_angle scaled (bts*bos*0.99*mincho_blob_size)
217     shifted plp;
218   z2=(0.5[x3,x4],1);
219   if l=0:
220     z3=(-0.5,0);
221     z4=(1.3,0);
222   else:
223     z3=(-1.3,0);
224     z4=(0.5,0);
225   fi;
226   serif:=((subpath (0.2,2) of (z4{up}..z2..{down}z3))-cycle) transformed t;
227   glstk[ngls]:=regenerate(serif);
228   ngls:=ngls+1;
229 endgroup;
230 enddef;
231
232 % standard serif codes:
233 % 1 - Latin left
234 % 2 - Latin right
235 % 3 - Latin left and right
236 % 4 - Mincho corner blob
237 % 5 - Mincho blob, pointy to ul
238 % 6 - Mincho blob, triangular
239 % 7 - Mincho blob, point to ll
240 % 8 - Mincho blob, point to l
241 % 9 - Mincho kanji triangle
242 % 10 - Mincho kanji half-ellipse
243
244 boolean tsu_do_serif[];
245
246 vardef tsu_serif.standard(expr bst,plp,dlp,l,bts,bos) =
247   if known tsu_do_serif[1]:
248     if tsu_do_serif[1] and (bst=1):
249       tsu_serif.latin.left(bst,plp,dlp,l,bts,bos);
250     fi;
251   fi;
252   if known tsu_do_serif[2]:
253     if tsu_do_serif[2] and (bst=2):
254       tsu_serif.latin.right(bst,plp,dlp,l,bts,bos);
255     fi;
256   fi;
257   if known tsu_do_serif[3]:
258     if tsu_do_serif[3] and (bst=3):
259       tsu_serif.latin.leftright(bst,plp,dlp,l,bts,bos);
260     fi;
261   fi;
262   if known tsu_do_serif[4]:

```

```

263   if tsu_do_serif[4] and (bst=4):
264       tsu_serif.mincho.corner(bst,plp,dlp,l,bts,bos);
265   fi;
266 fi;
267 if known tsu_do_serif[5]:
268     if tsu_do_serif[5] and (bst=5):
269         tsu_serif.mincho.ulpoint(bst,plp,dlp,l,bts,bos);
270     fi;
271 fi;
272 if known tsu_do_serif[6]:
273     if tsu_do_serif[6] and (bst=6):
274         tsu_serif.mincho.triangle(bst,plp,dlp,l,bts,bos);
275     fi;
276 fi;
277 if known tsu_do_serif[7]:
278     if tsu_do_serif[7] and (bst=7):
279         tsu_serif.mincho.llpoint(bst,plp,dlp,l,bts,bos);
280     fi;
281 fi;
282 if known tsu_do_serif[8]:
283     if tsu_do_serif[8] and (bst=8):
284         tsu_serif.mincho.lpoint(bst,plp,dlp,l,bts,bos);
285     fi;
286 fi;
287 if known tsu_do_serif[9]:
288     if tsu_do_serif[9] and (bst=9):
289         tsu_serif.mincho.ktriangle(bst,plp,dlp,l,bts,bos);
290     fi;
291 fi;
292 if known tsu_do_serif[10]:
293     if tsu_do_serif[10] and (bst=10):
294         tsu_serif.mincho.khellipse(bst,plp,dlp,l,bts,bos);
295     fi;
296 fi;
297 endif;
298
299 vardef tsu_serif.choose(expr bst,plp,dlp,l,bts,bos) =
300   tsu_serif.standard(bst,plp,dlp,l,bts,bos);
301 endif;

```