

作りましょう 0.11

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 0.11

Parametric Font Family
Proofs and pretty-printed
source code

Matthew Skala

mskala@ansuz.sooke.bc.ca

2021年7月1日 July 1, 2021

Proofs and pretty-printed source code for Tsukurimashou
Copyright © 2011–2021 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	19
	tsuku-kg.mp	20
	tsuku-mg.mp	21
	tsuku-mi.mp	22
	tsuku-ps.mp	24
	tsuku-bq.mp	25
	tsuku-dq.mp	26
	tsuku-el.mp	27
	tsuku-eq.mp	28
	tsuku-lw.mp	29
	intro.mp	30
	fntbase.mp	49
	General Library Functions	49
	Prefix And Suffix Handling	56
	A Module That Finds An Envelope Of A Path Drawn With A Pen	58
	Postscript Font Generation	70
	obstack.mp	92
	Object Stack Data	92
	Object Stack Methods	93
	frac-intro.mp	98
	latin-intro.mp	100
	accent.mp	104
	bcircle.mp	115
	bkencl.mp	118
	buildkanji.mp	125
	dakuten.mp	138
	enclosed.mp	139
	genjimon.mp	141
	hiragana.mp	153
	Hiragana Vowels	153
	Hiragana Kakikukeko/Gagigugego	158

Hiragana Sashisuseso/Zajizuzezo	163
Hiragana Tachitsuteto/Dajizudedo	169
Hiragana Naninuneno	174
Hiragana Hahifuheho/Babibubebo/Papipupepo	180
Hiragana Mamimumemo	185
Hiragana Yayuyo	191
Hiragana Rarirurero	193
Hiragana Wawiwewo/N/Iteration	199
iching.mp	207
katakana.mp	209
Katakana Vowels	209
Katakana Kakikukeko/Gagigugego	213
Katakana Sashisuseso/Zajizuzezo	218
Katakana Tachitsuteto/Dajizudedo	223
Katakana Naninuneno	228
Katakana Hahifuheho/Babibubebo/Papipupepo	233
Katakana Mamimumemo	239
Katakana Yayuyo	243
Katakana Rarirurero	246
Katakana Wawiwewo/N/Iteration	252
latin.mp	258
numerals.mp	347
ogonek.mp	358
punct.mp	370
serif.mp	412
 II Shared kyouiku kanji	 419
gradeone.mp	421
gradetwo.mp	493
gradethree.mp	600
gradefour.mp	683
gradefive.mp	765
gradesix.mp	817
 III Other shared kanji	 841
bottomrad.mp	843
leftrad.mp	847

radical.mp	857
rightrad.mp	903
toprad.mp	909
gradeeight.mp	917
gradenine.mp	987
gradeten.mp	1031
rare.mp	1032

IV U+0000 to U+0FFF 1109

tsuku-00.mp	1111
Ascii	1111
Latin-1 Extra Characters	1205
Accented Latin	1226
tsuku-01.mp	1284
Latin Extended A Uppercase	1284
Latin Extended A Lowercase	1339
Latin Extended A Other	1397
tsuku-02.mp	1399
Latin Extended B	1399
Spacing Modifier Letters	1400
tsuku-03.mp	1409
Combining Diacritical Marks	1409

V U+1000 to U+2FFF 1425

tsuku-20.mp	1427
General Punctuation	1427
tsuku-21.mp	1444
Symbols Required By Mes-1	1444
tsuku-24.mp	1450
Circled Numerals	1450
Circled Latin And Zero	1470
Inverted Circled Numerals	1523
Doubly Circled Numerals	1533
One More Inverted Circled Numeral	1543
tsuku-25.mp	1545
Geometric Shapes	1545
tsuku-26.mp	1547

I Ching	1547
tsuku-27.mp	1564
Inverted Circled Numerals	1564
tsuku-2e.mp	1575
Cjk Radicals Supplement	1575
tsuku-2f.mp	1616
Kangxi Radicals	1616
Ideographic Description Characters	1736

VI U+3000 to U+4DFF	1751
tsuku-30.mp	1753
Ideographic Symbols And Punctuation	1753
Hiragana	1773
Katakana	1818
tsuku-31.mp	1865
Phonetic Extensions For Ainu	1865
tsuku-32.mp	1882
Circled Numerals	1882
Circled Katakana	1912
tsuku-34.mp	1960
tsuku-35.mp	1961
tsuku-4d.mp	1962
I Ching	1962

VII U+4E00 to U+56FF	2027
tsuku-4e.mp	2029
tsuku-4f.mp	2062
tsuku-50.mp	2118
tsuku-51.mp	2135
tsuku-52.mp	2150
tsuku-53.mp	2197
tsuku-54.mp	2217
tsuku-55.mp	2226
tsuku-56.mp	2231

VIII	U+5700 to U+5EFF	2239
	tsuku-57.mp	2241
	tsuku-58.mp	2361
	tsuku-59.mp	2427
	tsuku-5a.mp	2442
	tsuku-5b.mp	2448
	tsuku-5c.mp	2502
	tsuku-5d.mp	2525
	tsuku-5e.mp	2534
IX	U+5F00 to U+65FF	2567
	tsuku-5f.mp	2569
	tsuku-60.mp	2634
	tsuku-61.mp	2763
	tsuku-62.mp	2815
	tsuku-63.mp	2834
	tsuku-64.mp	2849
	tsuku-65.mp	2852
X	U+6600 to U+69FF	2887
	tsuku-66.mp	2889
	tsuku-67.mp	2972
	tsuku-68.mp	3099
	tsuku-69.mp	3218
XI	U+6A00 to U+79FF	3317
	tsuku-6a.mp	3319
	tsuku-6b.mp	3381
	tsuku-6c.mp	3394
	tsuku-6d.mp	3444
	tsuku-6e.mp	3459
	tsuku-6f.mp	3475
	tsuku-70.mp	3483
	tsuku-71.mp	3489
	tsuku-72.mp	3498
	tsuku-73.mp	3515

tsuku-74.mp	3518
tsuku-75.mp	3522
tsuku-76.mp	3535
tsuku-77.mp	3558
tsuku-78.mp	3567
tsuku-79.mp	3572

XII U+7A00 to U+89FF	3625
tsuku-7a.mp	3627
tsuku-7b.mp	3677
tsuku-7c.mp	3692
tsuku-7d.mp	3698
tsuku-7e.mp	3809
tsuku-7f.mp	3829
tsuku-80.mp	3836
tsuku-81.mp	3850
tsuku-82.mp	3855
tsuku-83.mp	3887
tsuku-84.mp	3901
tsuku-85.mp	3906
tsuku-86.mp	3908
tsuku-87.mp	3910
tsuku-88.mp	3912
tsuku-89.mp	3943

XIII U+8A00 to U+9FFF	3957
tsuku-8a.mp	3959
tsuku-8b.mp	4051
tsuku-8c.mp	4086
tsuku-8d.mp	4106
tsuku-8e.mp	4110
tsuku-8f.mp	4114
tsuku-90.mp	4131
tsuku-91.mp	4157
tsuku-92.mp	4170
tsuku-93.mp	4177
tsuku-95.mp	4181

tsuku-96.mp	4245
tsuku-97.mp	4273
tsuku-98.mp	4277
tsuku-99.mp	4340
tsuku-9a.mp	4345
tsuku-9b.mp	4350
tsuku-9c.mp	4353
tsuku-9e.mp	4355
tsuku-9f.mp	4360

XIV U+A000 to U+10FFFF	4363
tsuku-f7.mp	4365
Latin Small Caps	4365
tsuku-f9.mp	4392
tsuku-ff.mp	4394
Full-Width Forms	4394
Half-Width Punctuation	4420
Half-Width Katakana	4423
tsuku-1f1.mp	4487
Squared Latin	4487
Inverse Circled Latin	4513
Inverse Squared Latin	4539
tsuku-200.mp	4566
tsuku-20a.mp	4567
tsuku-21c.mp	4568
tsuku-295.mp	4569
tsuku-f17.mp	4570
Combining Dots For I Ching	4570
Miscellaneous	4579
Tomoe Ornaments	4585
Heavy Metal Umlaut	4593
Genjimon	4610
tsuku-ff0.mp	4664
Fraction Numerators	4664
tsuku-ff1.mp	4671
Fraction Denominators	4671

XV Mandeubsida core	4673
hangul.mp	4675
Jamo Combining Operations	4677
jamo-basic.mp	4685
Filler Jamo	4685
Sios/Cieuc/Chieuch Family	4685
Kiyek	4688
Nieun	4689
Tikeut	4691
Rieul	4693
Mieum	4694
Pieup	4695
Sios	4696
Ieung	4696
Cieuc	4698
Chieuch	4698
Khieukh	4699
Thieuth	4700
Phieuph	4701
Hieuh	4703
Mixed Tails	4704
Vowels	4705
jamo-extra.mp	4714
Pansios	4717
Yesieung	4717
Yeorinhieuh	4718
Chitueum And Ceongchieum Variants	4720
Kapyeoun Variants	4721
hglxtb.mp	4722
Hangul Extension B	4722
Hangul Jungseong (Vowel) Jamo Extension B	4722
Hangul Jongseong (Tail) Jamo Extension B	4745
mande-bt.mp	4795
mande-do.mp	4796
mande-sm.mp	4797
hglpage.mp	4798
mande-ll.mp	4800
Hangul Choseong (Lead) Jamo	4800
Hangul Jungseong (Vowel) Jamo	4895

Hangul Jongseong (Tail) Jamo	4966
mande-3l.mp	5055
Hangul Compatibility Jamo	5055
mande-a9.mp	5149
Hangul Choseong (Lead) Jamo Extended A	5149
mande-ac.mp	5179

XVI Mandeubsida alternates	5233
mande-ff2.mp	5235
Hangul Jungseong (Vowel) Jamo	5235
Hangul Jungseong (Vowel) Jamo Extension B	5306
mande-ff3.mp	5330
Hangul Choseong (Lead) Jamo	5330
mande-ff4.mp	5455
Hangul Choseong (Lead) Jamo	5455
mande-ff5.mp	5580
Hangul Choseong (Lead) Jamo	5580
mande-ff6.mp	5705
Hangul Choseong (Lead) Jamo	5705
mande-ff7.mp	5830
Hangul Choseong (Lead) Jamo	5830
Hangul Choseong (Lead) Jamo Extended A	5925

XVII TsuIta	5955
tsuita-common.mp	5957
tsuita-at.mp	5968
tsuita-so.mp	5969
Additional Proofs	5971

XVIII Blackletter Lolita	6025
bll.mp	6027
bll-co.mp	6032
pentacross.mp	6033
Utilities For Pentagrams And Crosses	6033
bll-f5c.mp	6035
Pentagrams	6035

XIX	Kazoemashou	6055
	kazoe-se.mp	6057
	kazoe-ld4.mp	6060
	Math Bold	6060
	Math Bold Italic	6067
	Math Italic	6074

Volume I

Infrastructure

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


```

1 %
2 % Early shared code for Tsukurimashou
3 % Copyright (C) 2011, 2012, 2013 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 def whatever_xf =
72   begingroup
73     save newxf;
74     transform newxf;
75     newxf
76   endgroup
77 enddef;
78
79 

---


80

```

Font Parameter Defaults

```

81 % FONT PARAMETER DEFAULTS
82
83 % framework for brush definitions
84 transform tsu_brush_xf[];
85 numeric tsu_brush_shape[];
86 numeric tsu_brush_angle[];
87 numeric tsu_brush_min[];
88 numeric tsu_brush_max[];

```



```

89
90 def brletter = 0 enddef;
91 def bralternate = 1 enddef;
92 def brpunct = 2 enddef;
93
94 % basic brush definition
95 % style MUST set tsu_brush_xf.brletter;
96 tsu_brush_shape.brletter:=1.0;
97 tsu_brush_angle.brletter:=0;
98 tsu_brush_min.brletter:=0.5;
99 tsu_brush_max.brletter:=1.0;
100
101 % alternate brush
102 (0,0) transformed tsu_brush_xf.bralternate=(1,1);
103 (0,1) transformed tsu_brush_xf.bralternate=(1,1);
104 (1,0) transformed tsu_brush_xf.bralternate=(1,1);
105 tsu_brush_shape.bralternate:=1.0;
106 tsu_brush_angle.bralternate:=0;
107 tsu_brush_min.bralternate:=0.5;
108 tsu_brush_max.bralternate:=0.5;
109
110 % punctuation brush
111 (0,4) transformed tsu_brush_xf.brpunct=(0,0.5);
112 (1,1) transformed tsu_brush_xf.brpunct=(1,0.1);
113 (4,0) transformed tsu_brush_xf.brpunct=(4,0.5);
114 tsu_brush_shape.brpunct:=1.0;
115 tsu_brush_angle.brpunct:=0;
116 tsu_brush_min.brpunct:=0.1;
117 tsu_brush_max.brpunct:=0.5;
118
119 % size for punctuation
120 tsu_punct_size:=100;
121
122 % size the handakuten
123 handakuten_inner:=120;
124 handakuten_outer:=200;
125
126 % general shape tweaker
127 mincho:=0;
128
129 % slant during rescaling, for italics
130 rescale_slant:=0;
131
132 % control appearance of corners
133 boolean sharp_corners;
134 sharp_corners:=false;
135
136 % for naming the font

```

```

137 if unknown familyname:
138   string familyname,stylename;
139   familyname:="Tsukurimashou";
140   stylename:="";
141 fi;
142
143 % brush option override
144 def tsu_brush_opt(expr n,l) = nib(n)(l) enddef;
145
146 % bo_serif type; point lp; direction lp; brush tip size; brush code
147 vardef tsu_serif.choose(expr bst,plp,dlp,l,bts,bos,b) =
148 enddef;
149
150 % do "modern" width alternation
151 boolean do_alteration;
152 do_alteration:=false;
153
154 % handle outline mode for Genjimon
155 boolean genji_outline;
156 genji_outline:=false;
157
158 % prepare to detect proportional spacing
159 boolean is_proportional;
160 is_proportional:=false;
161
162 % prepare to detect blackletter
163 boolean is_blackletter;
164 is_blackletter:=false;
165
166 % prepare to detect fine IDCs
167 boolean fine_idcs;
168 fine_idcs:=false;
169
170 % prepare to detect italic hook shapes
171 boolean do_italic_hook;
172 do_italic_hook:=false;

```

tsuku-bk.mp

BK

```
1 %
2 % Tsukurimashou Bokukko
3 % Copyright (C) 2011, 2013 Matthew Skala
4 %
5-29 [Standard copyright notice]
30
31 % TSUKURIMASHOU BOKUKKO
32
33 input preintro.mp;
34
35 if stylename="": stylename="Bokukko"; fi;
36
37 mincho:=0.3;
38
39 (0,4) transformed tsu_brush_xf.brletter = (0.8,0.95);
40 (1,1) transformed tsu_brush_xf.brletter = (1.02,0.80);
41 (4,0) transformed tsu_brush_xf.brletter = (3.8,0.95);
42
43 tsu_brush_min.brletter:=0.80;
44 tsu_brush_max.brletter:=0.95;
45 tsu_brush_shape.brletter:=0.3;
46 tsu_brush_angle.brletter:=20;
47
48 tsu_brush_xf.brpunct:=whatever_xf;
49 (0,4) transformed tsu_brush_xf.brpunct = (0,0.60);
50 (1,1) transformed tsu_brush_xf.brpunct = (1,0.10);
51 (4,0) transformed tsu_brush_xf.brpunct = (4,0.60);
52
53 tsu_brush_min.brpunct:=0.10;
54 tsu_brush_max.brpunct:=0.60;
55 tsu_brush_shape.brpunct:=0.3;
56 tsu_brush_angle.brpunct:=20;
57
58 def tsu_brush_opt(expr n,l) = cut(n,rel 120)(l) enddef;
59 sharp_corners:=true;
60
61 genji_outline:=true;
62 genji_hw:=0.55;
63
64 input intro.mp;
```

tsuku-kg.mp

KG

```
1 %
2 % Tsukurimashou Kaku
3 % Copyright (C) 2011, 2013 Matthew Skala
4 %
5-29 [Standard copyright notice]
30
31 % TSUKURIMASHOU KAKU
32
33 input preintro.mp;
34
35 if stylename="": stylename="Kaku"; fi;
36
37 (0,4) transformed tsu_brush_xf.brletter = (4,0.75);
38 (1,1) transformed tsu_brush_xf.brletter = (1,0.62);
39 (4,0) transformed tsu_brush_xf.brletter = (0,0.75);
40
41 tsu_brush_min.brletter:=0.62;
42 tsu_brush_max.brletter:=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

```
1 %
2 % Tsukurimashou Maru
3 % Copyright (C) 2011, 2013 Matthew Skala
4 %
5-29 [Standard copyright notice]
30
31 % TSUKURIMASHOU MARU
32
33 input preintro.mp;
34
35 if stylename=="": stylename="Maru"; fi;
36
37 (0,4) transformed tsu_brush_xf.brletter = (4,0.74);
38 (1,1) transformed tsu_brush_xf.brletter = (1,0.65);
39 (4,0) transformed tsu_brush_xf.brletter = (0,0.74);
40
41 tsu_brush_min.brletter:=0.65;
42 tsu_brush_max.brletter:=0.74;
43
44 input intro.mp;
```

tsuku-mi.mp

MI

```
1 %
2 % Tsukurimashou Mincho
3 % Copyright (C) 2011, 2013 Matthew Skala
4 %
5-29 [Standard copyright notice]
30
31 % TSUKURIMASHOU MINCHO
32
33 input preintro.mp;
34
35 if stylename="": stylename="Mincho"; fi;
36
37 mincho:=1;
38
39 (0,4) transformed tsu_brush_xf.brletter = (0,0,1,1);
40 (1,1) transformed tsu_brush_xf.brletter = (1,2,0,35);
41 (4,0) transformed tsu_brush_xf.brletter = (4,8,1,1);
42
43 tsu_brush_min.brletter:=0.35;
44 tsu_brush_max.brletter:=1.05;
45 tsu_brush_shape.brletter:=0.38;
46 tsu_brush_angle.brletter:=1;
47
48 tsu_brush_xf.bralternate:=identity xyscaled (1,2,0) shifted (0,0,315);
49 tsu_brush_min.bralternate:=0.315;
50 tsu_brush_max.bralternate:=0.315;
51 tsu_brush_shape.bralternate:=1;
52 tsu_brush_angle.bralternate:=0;
53
54 tsu_brush_xf.brpunct:=whatever_xf;
55 (0,4) transformed tsu_brush_xf.brpunct = (0,0,60);
56 (1,1) transformed tsu_brush_xf.brpunct = (1,0,117);
57 (4,0) transformed tsu_brush_xf.brpunct = (4,0,60);
58
59 tsu_brush_min.brpunct:=0.117;
60 tsu_brush_max.brpunct:=0.60;
61 tsu_brush_shape.brpunct:=1;
62 tsu_brush_angle.brpunct:=1;
63
64 input serif.mp;
65
66 for i=1 upto 10:
67   tsu_do_serif[i]:=true;
68 endfor;
69
70 do_alteration:=true;
```

```
71  
72 genji_outline:=true;  
73 genji_hw:=0.2;  
74  
75 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, 2013 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.brletter:=
39   tsu_brush_xf.brletter transformed weight_xf;
40 tsu_brush_min.brletter:=
41   ypart ((0,tsu_brush_min.brletter) transformed weight_xf);
42 tsu_brush_max.brletter:=
43   ypart ((0,tsu_brush_max.brletter) transformed weight_xf);
44
45 tsu_brush_xf.bralternate:=tsu_brush_xf.bralternate yscaled (1.191/0.75);
46 tsu_brush_min.bralternate:=tsu_brush_min.bralternate*(1.191/0.75);
47 tsu_brush_max.bralternate:=tsu_brush_max.bralternate*(1.191/0.75);
48
49 tsu_brush_xf.brpunct:=tsu_brush_xf.brpunct yscaled 1.15;
50 tsu_brush_min.brpunct:=tsu_brush_min.brpunct*1.15;
51 tsu_brush_max.brpunct:=tsu_brush_max.brpunct*1.15;
52 tsu_punct_size:=120;
53
54 handakuten_outer:=260;
55
56 calc_mbrush_size;
```

tsuku-dq.mp

```
1 %
2 % Demibold font weight
3 % Copyright (C) 2012, 2013 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.brletter:=
39   tsu_brush_xf.brletter transformed weight_xf;
40 tsu_brush_min.brletter:=
41   ypart ((0,tsu_brush_min.brletter) transformed weight_xf);
42 tsu_brush_max.brletter:=
43   ypart ((0,tsu_brush_max.brletter) transformed weight_xf);
44
45 tsu_brush_xf.bralternate:=tsu_brush_xf.bralternate yscaled (0.945/0.75);
46 tsu_brush_min.bralternate:=tsu_brush_min.bralternate*(0.945/0.75);
47 tsu_brush_max.bralternate:=tsu_brush_max.bralternate*(0.945/0.75);
48
49 tsu_brush_xf.brpunct:=tsu_brush_xf.brpunct yscaled 1.15;
50 tsu_brush_min.brpunct:=tsu_brush_min.brpunct*1.15;
51 tsu_brush_max.brpunct:=tsu_brush_max.brpunct*1.15;
52 tsu_punct_size:=110;
53
54 handakuten_outer:=260;
55
56 calc_mbrush_size;
```

tsuku-el.mp

```
1 %
2 % Extra-Light font weight (Tenshi no Kami when added to Maru)
3 % Copyright (C) 2011, 2012, 2013 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.brletter:=
39   tsu_brush_xf.brletter transformed weight_xf;
40 tsu_brush_min.brletter:=
41   ypart ((0,tsu_brush_min.brletter) transformed weight_xf);
42 tsu_brush_max.brletter:=
43   ypart ((0,tsu_brush_max.brletter) transformed weight_xf);
44
45 tsu_brush_xf.bralternate:=tsu_brush_xf.bralternate yscaled (0.15/0.75);
46 tsu_brush_min.bralternate:=tsu_brush_min.bralternate*(0.15/0.75);
47 tsu_brush_max.bralternate:=tsu_brush_max.bralternate*(0.15/0.75);
48
49 tsu_brush_xf.brpunct:=tsu_brush_xf.brpunct yscaled 0.15;
50 tsu_brush_min.brpunct:=tsu_brush_min.brpunct*0.15;
51 tsu_brush_max.brpunct:=tsu_brush_max.brpunct*0.15;
52 tsu_punct_size:=80;
53
54 handakuten_inner:=170;
55
56 fine_idcs:=true;
57
58 calc_mbrush_size;
```

EL

tsuku-eq.mp

```
1 %
2 % Extra-Bold font weight (Anbiruteki when added to Maru)
3 % Copyright (C) 2011, 2012, 2013 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.brletter:=
39   tsu_brush_xf.brletter transformed weight_xf;
40 tsu_brush_min.brletter:=
41   ypart ((0,tsu_brush_min.brletter) transformed weight_xf);
42 tsu_brush_max.brletter:=
43   ypart ((0,tsu_brush_max.brletter) transformed weight_xf);
44
45 tsu_brush_xf.bralternate:=tsu_brush_xf.bralternate yscaled (1.5/0.75);
46 tsu_brush_min.bralternate:=tsu_brush_min.bralternate*(1.5/0.75);
47 tsu_brush_max.bralternate:=tsu_brush_max.bralternate*(1.5/0.75);
48
49 tsu_brush_xf.brpunct:=tsu_brush_xf.brpunct yscaled 1.15;
50 tsu_brush_min.brpunct:=tsu_brush_min.brpunct*1.15;
51 tsu_brush_max.brpunct:=tsu_brush_max.brpunct*1.15;
52 tsu_punct_size:=130;
53
54 handakuten_outer:=260;
55
56 calc_mbrush_size;
```

tsuku-lw.mp

```
1 %
2 % Light font weight
3 % Copyright (C) 2012, 2013 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.brletter:=
39   tsu_brush_xf.brletter transformed weight_xf;
40 tsu_brush_min.brletter:=
41   ypart ((0,tsu_brush_min.brletter) transformed weight_xf);
42 tsu_brush_max.brletter:=
43   ypart ((0,tsu_brush_max.brletter) transformed weight_xf);
44
45 tsu_brush_xf.bralternate:=tsu_brush_xf.bralternate yscaled (0.439/0.75);
46 tsu_brush_min.bralternate:=tsu_brush_min.bralternate*(0.439/0.75);
47 tsu_brush_max.bralternate:=tsu_brush_max.bralternate*(0.439/0.75);
48
49 tsu_brush_xf.brpunct:=tsu_brush_xf.brpunct yscaled 0.35;
50 tsu_brush_min.brpunct:=tsu_brush_min.brpunct*0.35;
51 tsu_brush_max.brpunct:=tsu_brush_max.brpunct*0.35;
52 tsu_punct_size:=90;
53
54 handakuten_inner:=170;
55
56 fine_idcs:=true;
57
58 calc_mbrush_size;
```

LW

intro.mp

```
1 %
2 % General shared code for Tsukurimashou
3 % Copyright (C) 2011, 2012, 2013, 2015, 2016, 2017, 2021 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     numeric proof_stroke_i;
76     proof_stroke_i:=1;
77     init_obstack;
78     string perl_structure;
79     perl_structure:="$structure{""&name&""}=["""&name&""";
80 endif;
81
82 def endtsuglyph =
83     if rescale_to.right>0:
84         fix_hsbw((rescale_to.left+rescale_to.right),0,0);
85     else:
86         fix_hsbw(0,0,0);
87     fi;
88 endglyph;
89 perl_structure:=perl_structure&"]";
90 write "PERL_STRUCTURE ""&perl_structure to "proof.prf";
91 if rescale_to.right>0:
92     write ("ENDGLYPH 0 ""&decimal (rescale_to.left+rescale_to.right))
93         to "proof.prf";
94 else:
95     write "ENDGLYPH -10 0" to "proof.prf";
96 fi;
97 sp:=0;
98 if extra_ticks>0:
99     errmessage "Unconsumed extra tick";
100 fi;
101 endif;
102
103 def tsu_brush_tip_size(expr l,q,b) =
104     begingroup
105         numeric y,yy;
106         y:=ypart (point l of q);
107         if y<tsu_brush_min[b]:
108             yy:=tsu_brush_min[b];
109         elseif y>tsu_brush_max[b]:
110             yy:=tsu_brush_max[b];
111         else:
112             yy:=y;
113         fi;
114     yy
115 endgroup
116 endif;
117
118 def tsu_brush_tip(expr l,p,q,bsi,is_start,is_end,b) =

```

```

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

```



```

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

```

```

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

```

```

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

```

```

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

```

```

359     wt=ypart (width_curve intersectionpoint
360         ((wf,infinity)-(wf,infinity)));
361     % find transformation
362     if wf>0:
363         (rescale_from.left+lf,rescale_from.bottom) transformed t=
364             (rescale_to.left+lt,rescale_to.bottom-rescale_skew);
365         (rescale_from.left+lf,rescale_from.top) transformed t=
366             (rescale_to.left+lt,rescale_to.top-rescale_skew);
367         (rescale_from.right-rf,rescale_from.bottom) transformed t=
368             (rescale_to.right-rt,rescale_to.bottom+rescale_skew);
369     else:
370         (rescale_from.left+lf,rescale_from.bottom) transformed t=
371             (rescale_to.left+lt,rescale_to.bottom);
372         (rescale_from.left+lf,rescale_from.top) transformed t=
373             (rescale_to.left+lt,rescale_to.top);
374         (rescale_from.left+lf+1,rescale_from.bottom) transformed t=
375             (rescale_to.left+lt+1,rescale_to.bottom);
376     fi;
377     pair cp;
378     transform st;
379     cp:=((rescale_to.left+rescale_to.right)/2,rescale_to.bottom);
380     cp transformed st=cp;
381     cp+(100,0) transformed st=cp+(100,0);
382     cp+(0,100) transformed st=cp+(rescale_slant/10,100);
383     t:=t transformed st;
384     endgroup; fi;
385     t
386 endgroup
387 enddef;
388
389 % solve the quadratic equation  $ax^2+bx+c=0$ , including pathological cases
390 vardef solve_quadratic(expr a,b,c) =
391     if abs(a)<0.00001:
392         if abs(b)<0.00001:
393             if abs(c)<0.00001:
394                 (0,whatever)
395             else:
396                 (whatever,whatever)
397             fi
398         else:
399             (-c/b,whatever)
400         fi
401     elseif abs(a)<abs(b)/500:
402         (whatever,whatever)
403     else:
404         save d;
405         numeric d;
406         d=b*b-4*a*c;

```

```

407   if d>0:
408       ((-b-sqrt(d),b+sqrt(d))/(2*a))
409   elseif abs(d)<0.00001:
410       (-b/(2*a),whatever)
411   else:
412       (whatever,whatever)
413   fi
414 fi
415 enddef;
416
417 % find the t-values of any inflection points of subpath (0,1) of p
418 vardef segment_inflections(expr p) =
419     save x,y,c;
420     numeric x[],y[],c[];
421
422     % normalize to prevent numerical misbehaviour
423     z10=(point 0 of p)+z22;
424     z11=(postcontrol 0 of p)+z22;
425     z12=(precontrol 1 of p)+z22;
426     z13=(point 1 of p)+z22;
427     z10+z11+z12+z13=(0,0);
428     c10=(abs(z10)+abs(z11)+abs(z12)+abs(z13))/4;
429     if c10<0.1:
430         c10:=0.1;
431     fi;
432     z0=z10/c10;
433     z1=z11/c10;
434     z2=z12/c10;
435     z3=z13/c10;
436
437     % abort if points are close enough to collinear
438     if (abs(x0*y1-x1*y0)<0.01) and (abs(x2*y3-x3*y2)<0.01):
439         (whatever,whatever)
440     else:
441
442         % find t-values at which |z'(t) cross z''(t)|=0
443         c2=y0*(-1*x1 +2*x2 -x3)
444             +y1*( x0 -3*x2 +2*x3)
445             +y2*(-2*x0 +3*x1 -x3)
446             +y3*( x0 -2*x1 +x2 );
447
448         c1=y0*( 2*x1 -3*x2 +x3)
449             +y1*(-2*x0 +3*x2 -x3)
450             +y2*( 3*x0 -3*x1 )
451             +y3*( -x0 +x1 );
452
453         c0=y0*( -x1 +x2)
454             +y1*( x0 -x2)

```

```

455     +y2*( -x0 +x1 );
456
457     z20=solve_quadratic(c2,c1,c0);
458
459     % filter and sort to find points properly within the segment
460     if known x20:
461         if (x20>0.01) and (x20<0.99):
462             x21=x20;
463             fi;
464         fi;
465     if known y20:
466         if (y20>0.01) and (y20<0.99):
467             y21=y20;
468             fi;
469         fi;
470     if known x21:
471         z21
472     else:
473         (y21,x21)
474     fi
475 fi
476 enddef;
477
478 % version of insert_nodes modified to *always* insert, which is needed
479 % to keep node numbers in sync on pen-size-control paths
480 vardef really_insert_nodes(expr p)(text t) =
481   save j_, p_, s_, t_; path p_; p_:=p;
482   t_:=0;
483   for i_:=t:
484     t_[incr t_]=arclength(subpath(0,i_ mod length(p_)) of p_);
485     if abs(t_[t_])<0.0001: t_[t_]:=0; fi;
486   endfor
487   for i_:=1 upto t_:
488     s_:=arctime t_[i_] of p_;
489     p_:=(subpath (0, s_) of p_) && (subpath (s_,length p_) of p_)
490     if cycle p_: & cycle fi;
491   endfor;
492   p_
493 enddef;
494
495 % render a single segment - pulled out to make it easier to override
496 def tsu_render_segment(expr i,p,q,b) =
497   default_nib:=fix_nib(obstackna.bosize[i]*tsu_brush_max[b],
498     obstackna.bosize[i]*tsu_brush_max[b]
499     *tsu_brush_shape[b],
500     tsu_brush_angle[b]);
501   path mytip[],glyph;
502   for l=0 step 1 until length(p):

```

```

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

```



```

551 endfor;
552 %
553 % main render
554 %
555 for i=1 upto sp-1: if obstacktype[i]=othook:
556     if obstackn[i]=hsmain__render:
557         scantokens obstacks[i];
558     fi;
559 fi; endfor;
560 begingroup
561     numeric i,j,k,l,s,t,si,ngls;
562     path bqi,p,q,glstk[];
563     ngls:=0;
564     for i=1 upto sp-1: if obstacktype[i]=otstroke:
565         if unknown obstackna.bobrush[i]:
566             obstackna.bobrush[i]:=brletter;
567         fi;
568         if (obstackna.bobrush[i]=bralternate) and not do_alteration:
569             obstackna.bobrush[i]:=brletter;
570         fi;
571 % message "suffix " & str i;
572     bqi:=obstackq[i] transformed tsu_brush_xf[obstackna.bobrush[i]];
573     s:=0;
574     for j=0 step 1 until (length obstackp[i])-1:
575         k:=j+1;
576 % message " j=" & decimal j & " thr " & decimal (xpart point j of bqi)
577 % & "/" & decimal (xpart point k of bqi);
578         if ((xpart point j of bqi)<1)
579             and ((xpart point k of bqi)>=1):
580 % message " START";
581             if (xpart point k of bqi)=1:
582                 s:=k;
583             else:
584                 s:=j+(xpart ((subpath (j,k) of bqi)
585                     intersectiontimes ((1,infinity)
586                         -(1,infinity))));
587             fi;
588         fi;
589         if (((xpart point j of bqi)>=1) and ((xpart point k of bqi)<1))
590             or (k=length obstackp[i]):
591 % message " END";
592             if (xpart point k of bqi)>=1:
593                 t:=k;
594             else:
595                 t:=j+(xpart ((subpath (j,k) of bqi)
596                     intersectiontimes ((1,infinity)
597                         -(1,infinity))));
598             fi;

```

```

599     if ((t-s)>0.02) and (obstackna.bosize[i]>0):
600         p:=subpath (s,t) of obstackp[i];
601         q:=subpath (s,t) of bq;
602         numeric ltime[];
603         ltime[0]:=s;
604         for l=1 step 1 until (length p)-1:
605             ltime[l]:=floor (s+l);
606         endfor;
607         ltime[length p]:=t;
608         l:=0;
609         forever:
610             exitif l=length p;
611             begingroup
612                 save x,y;
613                 numeric x[],y[];
614                 z10=segment_inflections(subpath (l,l+1) of p);
615                 if known x10:
616                     p:=really_insert_nodes(p)(l+x10);
617                     q:=really_insert_nodes(q)(l+x10);
618                     for ll=length p step -1 until l+1:
619                         ltime[ll]:=ltime[ll-1];
620                     endfor;
621                     ltime[l+1]:=x10[ltime[l],ltime[l+2]];
622                 else:
623                     z0=(point l of p)/100;
624                     z1=(postcontrol l of p)/100;
625                     z2=(precontrol (l+1) of p)/100;
626                     z3=(point (l+1) of p)/100;
627                     if if abs(z2-z1)>0.1: ((z1-z0) dotprod (z3-z2))
628                         /((z2-z1) dotprod (z2-z1))
629                         else: 1 fi<0.5:
630                         p:=really_insert_nodes(p)(l+0.5);
631                         q:=really_insert_nodes(q)(l+0.5);
632                         for ll=length p step -1 until l+1:
633                             ltime[ll]:=ltime[ll-1];
634                         endfor;
635                         ltime[l+1]:=0.5[ltime[l],ltime[l+2]];
636                     else:
637                         l:=l+1;
638                     fi;
639                 fi;
640             endgroup;
641         endfor;
642         write ("SEGMENT "&(decimal proof_stroke_i)&" "&
643             (decimal s)&" "&(decimal t))
644         to "proof.prf";
645     for lcbj=0 upto length p:
646         write ("POINT "&(decimal proof_stroke_i)&" "

```

```

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

```

```

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

```

```

743     topanchor:=i;
744     for j:=sp-1 downto i+1:
745         if obstacktype[j]=otanchor:
746             if obstackn[j]=obstackn[i]:
747                 topanchor:=j;
748             fi;
749         fi;
750     exitif topanchor<>i;
751 endfor;
752 if topanchor=i:
753     write ("ANCHOR "&
754         (decimal obstackn[i]))&" "&
755         (decimal xpart ((-35,0) transformed obstackt[i]))&" "&
756         (decimal ypart ((-35,0) transformed obstackt[i]))&" "&
757         (decimal xpart ((35,0) transformed obstackt[i]))&" "&
758         (decimal ypart ((35,0) transformed obstackt[i]))&" "&
759         (decimal xpart ((0,-35) transformed obstackt[i]))&" "&
760         (decimal ypart ((0,-35) transformed obstackt[i]))&" "&
761         (decimal xpart ((0,35) transformed obstackt[i]))&" "&
762         (decimal ypart ((0,35) transformed obstackt[i]))&
763         to "proof.prf";
764     fi;
765 endgroup;
766 fi;
767 endfor;
768 endgroup;
769 enddef;
770
771 % the usual case - just render it without fitting into a circle
772 def tsu_render =
773     tsu_render_in_circle(identity scaled -1);
774 enddef;
775
776 transform tsu_xf.smallkana;
777
778 tsu_xf.smallkana = identity shifted (-500,0) scaled 5.5/8 shifted (500,0);
779
780 def tsu_xform(expr xform)(text curves) =
781     begingroup
782         save txfsp,zc,zs;
783         txfsp:=sp;
784         curves;
785         numeric zs;
786         zs:=abs(((0,0) transformed xform)
787             -((1,0) transformed xform))
788             *abs(((0,0) transformed xform)
789             -((0,1) transformed xform));
790         size_scale:=zs**0.16667;

```

```

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

```

```

839 boolean accent_has_default[];
840 max_accent_seen:=0;
841
842 def tsu_default_anchor(expr aindex,avalue) =
843   if numeric avalue:
844     write ("DEFAULTANCHOR "&(decimal aindex)&" FALSE") to "proof.prf";
845     accent_has_default[aindex]:=false;
846   elseif transform avalue:
847     write ("DEFAULTANCHOR "&
848           (decimal aindex)&" "&
849           (decimal xpart ((0,0) transformed avalue
850                           transformed tsu_rescale_xform))&" "&
851           (decimal ypart ((0,0) transformed avalue
852                           transformed tsu_rescale_xform)))
853           to "proof.prf";
854     accent_has_default[aindex]:=true;
855   elseif pair avalue:
856     write ("DEFAULTANCHOR "&
857           (decimal aindex)&" "&
858           (decimal xpart (avalue transformed tsu_rescale_xform))&" "&
859           (decimal ypart (avalue transformed tsu_rescale_xform))
860           to "proof.prf";
861     accent_has_default[aindex]:=true;
862   fi;
863   if aindex>max_accent_seen:
864     max_accent_seen:=aindex;
865   fi;
866 enddef;
867
868
869
870 % figure out size of brush
871 vardef calc_mbrush_size =
872   numeric mbrush_width,mbrush_height,alternate_adjust;
873   (mbrush_width,mbrush_height)=urcorner (
874     fullcircle xscaled (tsu_brush_max.brletter*100)
875     yscaled (tsu_brush_max.brletter*tsu_brush_shape.brletter*100)
876     rotated tsu_brush_angle.brletter
877   );
878   alternate_adjust:=abs(mbrush_height-mbrush_width);
879   if tsu_brush_max.brletter>0.75:
880     serif_size:=1.5;
881   else:
882     serif_size:=1.5*((tsu_brush_max.brletter/0.75)**(1/3));
883   fi;
884   mincho_blob_size:=sqrt(tsu_brush_max.brletter/0.75);
885   (sbrush_width,sbrush_height)=urcorner (
886     fullcircle yscaled tsu_brush_shape.brletter

```

```

887     rotated tsu_brush_angle.brletter
888 );
889 if sbrush_width>sbrush_height:
890     sbrush_long:=sbrush_width;
891     sbrush_short:=sbrush_height;
892 else:
893     sbrush_short:=sbrush_width;
894     sbrush_long:=sbrush_height;
895 fi;
896 endif;
897
898 calc_mbrush_size;
899
900 _____
901
902 extra_ticks:=0;
903
904 def extra_tick =
905     extra_ticks:=extra_ticks+1;
906 endif;
907
908 def consume_extra_tick =
909     if extra_ticks>0:
910         begingroup
911             extra_ticks:=extra_ticks-1;
912             true
913         endgroup
914     else:
915         false
916     fi
917 endif;

```

INTR

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
```

FNTB

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.mf
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 |postdir| and |predir| given below is
89 % based on the following observation, being the consequence
90 % of l'H\opital's rule: consider a B\ezier segment
91 % |a .. controls b and c .. d|; 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)
155   % doesn't work for "straight lines"
156 enddef;
157 vardef pretension expr t of p = % ditto
158   save q_; path q_;
159   q_:=point t-1 of p {direction t-1 of p} .. {direction t of p} point t of p;
160   length(precontrol 1 of q_ - point 1 of q_)/
161   length(precontrol t of p - point t of p)
162   % doesn't work for "straight lines"
163 enddef;
164
165 % The two macros below, |path_eq| and |inside| macros, might have been
166 % primitives. The macro |path_eq| is obvious; |a inside b| returns true
167 % if the bounding box of |a| is inside the bounding box of |b|, which
168 % may be misleading; think, for example of:
169 % |fullcircle inside unitsquare shifted (-1/2,-1/2) scaled .9 rotated 45|.
170 % For most curves occurring in fonts, however, one can safely infer
171 % that if |a inside b| holds, then |a| is inside |b|.
172 vardef path_eq(expr a,b)=
173   save i_,l_,r_; boolean r_;
174   r_:=((length(a)=length(b)) and (cycle a= cycle b));
175   if r_:
176     i_:=0; l_:=length(a) if cycle a: -1 fi;
177     forever:
178       r_:=((point i_ of a = point i_ of b); exitif not r_);
179       r_:=((precontrol i_ of a = precontrol i_ of b); exitif not r_);
180       r_:=((postcontrol i_ of a = postcontrol i_ of b); exitif not r_);
181       exitif incr i_>l_;
182     endfor fi
183   r_
184 enddef;
185
186 tertiarydef a inside b =
187   if path a: % |and path b|

```

```

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

```

```

236 clearxy; save d_; d_:=sqrt(length(b-c));
237 z0=a/d_; z1=b/d_; z2=c/d_;
238 (x2-x1)*(y1-y0)-(x1-x0)*(y2-y1)
239 endif;
240
241 % The idea of calculation of a turning angle
242 % between two vectors, employed in the definition of the function
243 % 'turn_ang,' is based on the following observation:
244 vardef turn_ang(expr za,zb) =
245   if (abs(za)>=1/1000) and (abs(zb)>=1/1000): % lepl may be not enough
246     angle(unitvector(za) zscaled
247       (unitvector(zb) reflectedabout (origin,right)))
248   else: whatever fi
249 endif;
250
251 % A Boolean function 'is_line1' checks whether a given B\ezier segment
252 % is a straight line. For large segments (fonts) it makes sense to specify
253 % a numerical parameter is_line_off>=0; it defines a maximal acceptable
254 % distance of the control points of a B\ezier arc from its secant
255 % (which corresponds to the distance between the arc and the secant
256 % circa |3/4is_line_off| for a symmetric, inflexionless arcs).
257 vardef is_line(expr B) =
258   save r_; boolean r_;
259   if known is_line_off:
260     save a_;
261     a_:=length((point 1 of B)-(point 0 of B));
262     r_:=(-a_+arclength(B))<=(a_/infinity);
263     if r_:
264       r_:=((is_line_off>=abs(point_line_dist(
265         postcontrol 0 of B, point 0 of B, point 1 of B))) and
266         (is_line_off>=abs(point_line_dist(
267         precontrol 1 of B, point 0 of B, point 1 of B)))));
268     fi
269   else: % backward compatibility
270     save a_,b_,c_,d_;
271     a_:=length((point 1 of B)-(point 0 of B));
272     b_:=length((postcontrol 0 of B)-(point 0 of B));
273     c_:=length((precontrol 1 of B)-(postcontrol 0 of B));
274     d_:=length((point 1 of B)-(precontrol 1 of B));
275     r_:=(-a_+b_+c_+d_ <= a_/infinity);
276   fi
277   r_
278 endif;
279
280 % Abbreviations for a few simple yet useful phrases
281 def xyscaled primary p = xscaled xpart(p) yscaled ypart(p) endif;
282 def yxscaled primary p = yscaled xpart(p) xscaled ypart(p) endif;
283

```

```

284 % The macro |insert_nodes| inserts additional nodes at given non-integer
285 % non-repeating times |t| into a given path |p|.
286 % The code would be a bit longer without 'arclength' and 'arctime'.
287 % The macro can be useful in some cases in the context of finding
288 % the envelopes of pen-stroked paths (avoiding inflection
289 % points—see below).
290 vardef insert_nodes(expr p)(text t) =
291   save j_, p_, s_, t_; path p_; p_:=p;
292   t_:=0;
293   for i_:=t:
294     if round(i_)<>i_: % ignore integer times
295       t_[incr t_]=arclength(subpath(0,i_ mod length(p_) of p_);
296     fi
297   endfor
298   for i_:=1 upto t_:
299     s_:=arctime t_[i_] of p_;
300     if abs(round(s_)-s_)>eps: % ignore repeating times; is |eps| OK?
301       p_:=(subpath (0, s_) of p_) && (subpath (s_,length p_) of p_)
302       if cycle p_: & cycle fi;
303     fi
304   endfor;
305   p_
306 enddef;
307
308 % get rid of degeneracies
309 def regenerate(expr p) =
310   if (xpart urcorner p-xpart ulcorner p<5)
311     and (ypart ulcorner p-ypart llcorner p<5):
312     p
313   else:
314     begingroup
315       save q;
316       path q;
317       for t=1 step 1 until length p:
318         if abs((point t of p)-(point (t-1) of p))>3:
319           if unknown q:
320             q:=subpath (t-1,t) of p;
321           elseif length(q)=1:
322             q:=(point 0 of q)..
323               controls (postcontrol 0 of q) and (precontrol 1 of q)..
324               (0.5[point 1 of q,point t-1 of p])..
325               controls (postcontrol t-1 of p) and (precontrol t of p)..
326               (point t of p);
327           else:
328             q:=(subpath (0,length(q)-1) of q)..
329               controls (postcontrol length(q)-1 of q)
330                 and (precontrol length(q) of q)..
331               (0.5[point length(q) of q,point t-1 of p])..

```

```

332             controls (postcontrol t-1 of p) and (precontrol t of p)..
333             (point t of p);
334         fi;
335     fi;
336 endfor;
337 if cycle p:
338     q:=subpath (0,length(q)-1) of q..
339     controls (postcontrol length(q)-1 of q)
340     and (precontrol length(q) of q)..
341     cycle;
342 fi;
343 q
344 endgroup
345 fi
346 enddef;
347
348 % like Fill, but doesn't complain about turning number
349 def dangerousFill text glist =
350 begingroup
351   save h_; path h_;
352   for g_:=glist:
353     h_:=g_ start.default; % JMN's suggestion
354     if glyph_usage div store = 1: % storing
355       glyph_stored.glyph_name[incr glyph_stored.glyph_name.num]=h_;
356     fi
357     glyph_list[incr glyph_list.num]:=round_node_values(h_ italicized);
358     update_glyph_bb(glyph_list[glyph_list.num]);
359   endfor;
360 endgroup
361 enddef;
362
363
364

```

FNTB

Prefix And Suffix Handling

```

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

```



```

377 % suffix munging
378
379 def store_prec_obj = store_prec_obj__ whatever enddef;
380 primarydef a store_prec_obj__ b = hide(def prec_obj = a enddef) enddef;
381
382 % primarydef a sub b =
383 % if path a: (pos_subpath b of a) elseif string a: (substring b of a) fi
384 % enddef;
385
386 def node = store_prec_obj node__ enddef;
387 vardef node_@# primary a =
388   if str @#="x": xpart(point a of prec_obj)
389   elseif str @#="y": ypart(point a of prec_obj)
390   elseif str @#="": point a of prec_obj
391   else:
392     errhelp
393     "The operator 'node' works only with 'x', 'y' or an empty suffixes";
394     errmessage "PX: improper usage of 'node'";
395   fi
396 enddef;
397
398 def first suffix $ =
399   if str $="at": % moves the first point of a path to a specified location
400     store_prec_obj prec_obj shifted -(point 0 of prec_obj) shifted
401   else: node$(0) fi
402 enddef;
403 def last suffix $ =
404   if str $="at": % moves the last point of a path to a specified location
405     store_prec_obj prec_obj shifted
406     -(point if cycle prec_obj: 0 else: infinity fi of prec_obj) shifted
407   else: node$(if cycle prec_obj: 0 else: infinity fi) fi
408 enddef;
409
410 % Neat macros excerpted from John D. Hobby's boxes.mp macro package
411
412 % Find the length of the prefix of string |s| for which |cond| is true for
413 % each character |c| of the prefix
414 vardef genericize_prefix(expr s)(text cond) =
415   save i_, c_; string c_;
416   i_ = 0;
417   forever:
418     c_ := substring (i_,i_+1) of s;
419     exitunless cond; exitif incr i_=length s;
420   endfor
421   i_
422 enddef;
423
424 % Take a string returned by the |str| operator and return the same string

```

```

425 % with explicit numeric subscripts replaced by generic subscript symbols []:
426 vardef genericize(expr s) =
427   save res_, s_, l_; string res_, s_;
428   res_=""; % result so far
429   s_ =s; % left to process
430   forever: exitif s_="";
431   l_:=genericize_prefix(s_, (c_<>"[") and ((c_<"0") or (c_>"9")));
432   res_:=res_ & substring (0,l_) of s_;
433   s_:=substring (l_,infinity) of s_;
434   if s_<>"":
435     res_ := res_ & "[";
436     l_ :=if s_>="[" then 1+genericize_prefix(s_, c_<>"]")
437       else: genericize_prefix(s_, (c_="") or ("0"<=c_) and (c_<="9")) fi;
438     s_:=substring(l_,infinity) of s_;
439   fi
440 endfor
441 res_
442 enddef;
443
444
445

```

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

```

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

```

```

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

```

```

516 {direction (directiontime d of p_) of fullcircle}...
517 endfor cycle) xscaled x_diam yscaled y_diam rotated rot_angle unslant_stroke
518 enddef;
519
520 % Arcs of a pen shorter than |ignore_nib_limit| will be joined together
521 % to form larger ones. Remember to adjust the parameter |ignore_nib_limit|
522 % if the size of |default_nib| is significantly changed.
523 newinternal ignore_nib_limit; ignore_nib_limit:=1;
524
525 path default_nib;
526 default_nib:=fix_nib(50,50,0); % 100 times as large as a default plain pen
527
528 newinternal default_elongation, default_join, default_cap;
529 default_elongation:=1/2;
530 default_join:=1;
531 % 0 – tip, default elongation used
532 % 1 – pen join, default elongation ignored
533 % 2 – tip, default elongation ignored, elongation=0 used
534 default_cap:=1;
535 % 0 – cut 90 rel
536 % 1 – pen end
537
538 % |tangent_point|, |pen_join|, |pen_stroke_edge_|, and |pen_stroke_edge|
539 % are auxiliary macros, exploited by the main macro, i.e., |pen_stroke|.
540 vardef tangent_point(expr d,nib) = % |d| – direction of pen movement
541   save a_;
542   point if cycle nib: (directiontime d of nib) else:
543     if abs(d)<0.005:
544       1/2
545     else:
546       hide (a_:=turn_ang(d,(point 1 of nib)-(point 0 of nib)))
547       if abs(a_ mod 180)<.1: 1/2 % emergency
548       elseif a_<0: 0 else: 1 fi
549     fi
550   fi of nib
551 enddef;
552
553 vardef pen_join(expr a,b,c,nib)=
554   % deleting superfluous nodes is based on the |arclength| operation
555   % which, obviously, is not preserved after slanting, but let's hope
556   % it does not matter (too much)
557   save t_, m_, m__, ta_, tb_, p_; path p_;
558   m_:=infinity; % will be the minimal length of |nib|'s segment
559   for t_:=0 upto 1/2length(nib)-1:
560     m__:=arclength(subpath(t_,t_+1) of nib);
561     if m__<m_: m_:=m__; fi
562   endfor
563   if m_<ignore_nib_limit:

```

```

564 message "PX: the shortest nib segment < ignore_nib_limit (" &
565     decimal(m__) & " < " & decimal(ignore_nib_limit) & ")";
566 fi
567 p__=nib shifted c;
568 if cycle nib:
569     ta_=directiontime a of p__; tb_=directiontime b of p__;
570     p__:=pos_subpath(ta_,tb_) of p__;
571     if arclength(p__)>ignore_nib_limit:
572         for i_:=0,0:
573             p__:=reverse p__; % short segments may appear at both ends
574             if length(p__)>1: % optimization
575                 if arclength(subpath (0,1) of p__)<1/4ignore_nib_limit:
576                     % cf. the comment concerning |1/4ignore_nib_limit| in
577                     % |pen_stroke_edgel below
578                     p__:=(point 0 of p__) .. controls (postcontrol 1 of p__) and
579                     (precontrol 2 of p__) .. subpath (2,infinity) of p__;
580             fi
581         fi
582     endfor
583 else:
584     p__:=(point 0 of p__){a}...{b}(point length(p__) of p__);
585 fi
586 else: % razor nib
587     p__:=tangent_point(a,p__)-tangent_point(b,p__);
588 fi
589 p__
590 enddef;
591
592 % The finding of a pen envelope for a given B\ezier segment,
593 % defined by nodes |a|, |b|, |c|, and |d|, begins with
594 % the placing the pen at the ends of the B\ezier segment
595 % (i.e., at the points |a|, |d|) and finding the corresponding points
596 % |a'| and |d'| where the pen outline is parrallel to the direction
597 % of the original path at these points. Then, the outline is constructed.
598 % For |pen_stroke_method=0| (default), the envelope segment is constructed
599 % by setting the beginning and final directions (optionally, the direction
600 % at a given node can be ignored); for |pen_stroke_method=1| or 2
601 % an alternative (more elaborate) procedure is involved which explicitly
602 % computes control nodes |b'| and |c'| of the resulting path basing on
603 % a heuristic assumption that
604 %  $||length(b'-a')/length(b-a)|| \approx$ 
605 %  $||length(c'-d')/length(c-d)|| \approx$ 
606 %  $||length(a'-d')/length(a-d)||$  break
607 % The default method never produce concave edges because the operator |...|
608 % is used always; the alternative methods employs the operator
609 % |force_convex_edgel instead; for |pen_stroke_method=1| the convex edges
610 % are forced (i.e, inflexion points are being removed),
611 % for |pen_stroke_method=2| no forcing of convex edges takes place.

```

```

612 vardef extrapoline expr t of B = % the result may be not a single segment
613   save l_, t_;
614   (t_.a,t_.b)=t; % |0<=ta_<tb_<=1|!
615   l_:=arclength(B)/(t_.b-t_.a); l_.a=l_*t_.a; l_.b=l_*(1-t_.b);
616   if t_.a>0: ((point 0 of B) - l_.a*(upostdir 0 of B))- fi
617   B
618   if t_.b<1: - ((point 1 of B) + l_.b*(upredir 1 of B)) fi
619 enddef;
620
621 vardef force_convex_edge(expr za, zb, zc, zd) =
622   save a_, b_, c_, d_, z_;
623   a_:=length(zd-za); b_:=length(zb-za); c_:=length(zc-zb); d_:=length(zd-zc);
624   if (-a_+b_+c_+d_ > a_/infinity):
625     if pen_stroke_method=2:
626       za .. controls zb and zc .. zd
627     else:
628       if (a_>0.01) and (b_>0.01) and (c_>0.01) and (d_>0.01): % no degeneration
629         a_:=signum((za-zd) rotated -90 dotnorm (zb-za));
630         b_:=signum((zb-za) rotated -90 dotnorm (zc-zb));
631         c_:=signum((zc-zb) rotated -90 dotnorm (zd-zc));
632         d_:=signum((zd-zc) rotated -90 dotnorm (za-zd));
633         if ((a_<>b_) or (b_<>c_)) and (a_=d_):
634           numeric b_, c_; pair z_;
635           z_=b_[za,zb]=c_[zd,zc];
636           za .. controls
637             if b_<1: z_ else: zb fi and if c_<1: z_ else: zc fi
638             .. zd
639         else:
640           za .. controls zb and zc .. zd
641         fi
642       else:
643         za .. controls zb and zc .. zd
644       fi
645     fi
646   else:
647     za - zd
648   fi
649 enddef;
650
651 vardef pen_stroke_edge(expr b,b_nib,e_nib) = % |b| - B\ezier segment
652   save pa_,pb_,qa_,qb_,ra_,rb_,sa_,sb_;
653   pair pa_,pb_,qa_,qb_,ra_,rb_,sa_,sb_;
654   pa_:=point 0 of b; ra_=(postcontrol 0 of b)-pa_; sa_:=postdir 0 of b;
655   pb_:=point 1 of b; rb_=(precontrol 1 of b)-pb_; sb_:=predir 1 of b;
656   qa_:=pa_ + tangent_point(sa_, b_nib);
657   qb_:=pb_ + tangent_point(sb_, e_nib);
658   if pen_stroke_method=0:
659     qa_ {sa_} ... {sb_} qb_

```

```

660 elseif (pen_stroke_method=1) or (pen_stroke_method=2):
661   save lp_,lq_; lp_=length(pb_-pa_); lq_=length(qb_-qa_);
662   if 2lp_<lq_: % heuresis – too close nodes
663     qa_ {sa_} ... {sb_} qb_
664   else:
665     force_convex_edge(qa_, qa_+lq_/lp_*ra_, qb_+lq_/lp_*rb_, qb_)
666   fi
667 else:
668   errhelp "Only the values 0,1 and 2 for 'pen_stroke_method' " &
669     "are admissible. Better stop now.";
670   errmessage "PX: unknown pen stroke method (" &
671     decimal(pen_stroke_method) & ")";
672 fi
673 enddef;
674
675 vardef pen_stroke_edge@#(expr p) =
676   save e_,l_,i_,i__; path e_[\];
677   l_:=length(p);
678   for i_:=0 upto l_-1:
679     e_[i_]=pen_stroke_edge(subpath (i_,i_+1) of p,
680       % |local_nib_@#(i_),local_nib_@#(i_+1)| % a nasty bug removed 20.08.2009
681       local_nib_@#(i_),local_nib_@#((i_+1) if cycle p: mod l_ fi));
682   endfor
683   for i_:=0 upto l_ if cycle p: -1 else: -2 fi:
684     i__:= (i_+1) mod l_;
685     save t_;
686     t_:=turn_ang(predir 1 of e_[i_], postdir 0 of e_[i__]);
687     if if known t_: abs(t_)>1 else: false fi:
688       save t_; (t_.a,t_.b)=e_[i_] intersectiontimes e_[i__];
689       if t_.a>0:
690         e_[i_]:=subpath (0,t_.a) of e_[i_];
691         e_[i__]:=subpath (t_.b,l) of e_[i__];
692       elseif known local_tip_@#(i__):
693         save tx_, ty_, b_, b__, ei_, ei__; path ei_, ei__, ei_[], ei__[];
694         (tx_,ty_)=local_tip_@#(i__);
695         ei_:=if is_line(e_[i_]):
696           (point 0 of e_[i_])–
697           (1/abs(tx_))[point 0 of e_[i_], point 1 of e_[i_] ]
698         elseif tx_<0: hide(b_:=1) extrapoline (0,abs(tx_)) of e_[i_]
699         else: extrapolate (0,abs(tx_)) of e_[i_] fi;
700         ei__:=if is_line(e_[i__]):
701           (1/(1-abs(ty_)))[point 1 of e_[i__], point 0 of e_[i__] ] –
702           point 1 of e_[i__]
703         elseif ty_<0: hide(b__:=1) extrapoline (abs(ty_),1) of e_[i__]
704         else: extrapolate (abs(ty_),1) of e_[i__] fi;
705 % clumsy HEURESIS (choosing an optimal intersection point, if there are
706 % more intersections):
707       save t_; (t_.a,length(ei__)-t_.b)=ei_ intersectiontimes reverse ei__;

```

```

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

```



```

756 % it is also adequate for |pen_stroke_method=1|:
757 -
758 else:
759   if pen_stroke_method=0:
760     if not ignore_dir(i_): {postdir 0 of e_[i_]} fi ...
761     if not ignore_dir(i_+1): {predir 1 of e_[i_]} fi
762     elseif (pen_stroke_method=1) or (pen_stroke_method=2):
763       .. controls (postcontrol 0 of e_[i_]) and (precontrol 1 of e_[i_]) ..
764     fi
765   fi
766 endfor
767 if cycle p: cycle else: (point 1 of e_[l_-1]) fi
768 enddef;
769 newinternal pen_stroke_method;
770
771 % Macro |pen_stroke| performs an operation known as “expanding stroke”;
772 % we’ll call the result of the operation a “pen envelope” (for
773 % a given path). The macro has one optional parameter, |lopts| (|text|),
774 % and two obligatory ones: input path |p| (|expr|)
775 % and a |result| (|suffix|). A user has an access to subpaths of the
776 % envelope, namely: |result.r| is the right edge of the envelope,
777 % |result.l|—its left edge, |result.b|—is a fragment of the pen outline
778 % joining left and right edge of the envelope at the beginning
779 % node of the path, |result.e|—is a similar fragment at the ending
780 % node of the path (see the picture below). If the path |p|
781 % is cyclic, then |result.e| and |result.b| are undefined,
782 % otherwise the variable |result| contains additionally the complete
783 % expanded stroke.
784
785 % For finding an envelope, a default path (|default_nib|, returned
786 % by |fix_nib|) is used except nodes for which the parameter |lopts|
787 % sets another pen. Mastering the usage of the parameter |lopts| allows
788 % a user to achieve nontrivial effects. The parameter |lopts| is a list
789 % (space-separated or semicolon-separated) of the following
790 % operators: (1) |nib|, (2) |cut|, (3) |tip|, and (4) |ignore_directions|.
791
792 % Ad 1. The macro |nib| has two parameters:
793 % |nib|(pen)(list_of_nodes), where “pen” is a path returned by
794 % macro |fix_nib|, and “list_of_nodes” contains comma-separated numbers
795 % (times) of the nodes of the path |p| at which a given pen is to be
796 % used. If needed, the outline is complemented at corner nodes
797 % with a fragment of a pen path. Such a join corresponds to the setting
798 % |linejoin:=rounded| in \MP{.}. If the path |p| is non-cyclic,
799 % its ends are also complemented with appropriate fragments of a pen path
800 % (the setting |linecap:=rounded|). Such a method of joining is also applied
801 % by |pen_stroke| to nodes not mentioned in the parameter |lopts|.
802 % The result of the following statement
803 % \LINE{descriptioncomments

```

```

804 % |pen_stroke(nib(default_nib xyscaled (1,2))(infinity))(p)(q)|
805 % \unskip}
806 % \descriptioncomments
807 % that changes the pen at the last node of the path,
808 % is shown in the following picture:
809 % \LINE{\epsfbox{\illusname.110}}
810
811 % Ad 2. The call of the macro |cut| has the form: |cut|(angle,
812 % pen)(list_of_nodes) or |cut|(pen, angle)(list_of_nodes),
813 % where “pen” and “list_of_nodes” are defined as
814 % previously. The pen parameter can be omitted which means using a default
815 % pen (|default_nib|). The macro replaces a default pen with a special
816 % “razor” pen at specified nodes. More precisely, it is a projection of a
817 % given pen in the direction of the path |p| at a given node onto a
818 % straight line going through this node under the angle specified in the
819 % respective parameter of the macro. Uf\f\f\dots\ The angle of the straight
820 % line can be defined either absolutely (with respect to the axis |x|)
821 % or—by adding a prefix ‘|rell|’—relatively to the direction of the path
822 % at a given node. From the point of view of a user, the result of the
823 % macro |cut| is “cutting” the expanded stroke with a straight
824 % line. This operation is particularly useful at the ends of a path and
825 % corresponds to the setting |linecap:=butt| in \MP{, except that in \MP{
826 % one cannot specify angles. The result of the statement
827 % \LINE{\descriptioncomments
828 % |pen_stroke(cut(45)(0)|
829 % |cut(default_nib xyscaled (1,2), rel 90)(infinity))(p)(q)|
830 % \unskip}
831 % \descriptioncomments
832 % that cuts both ends and, moreover, changes a pen
833 % at the ending node is shown in the figure below
834 % (at the beginning node, the absolute angle of 45 degrees is specified,
835 % at the ending one—the relative angle of 90 degrees):
836
837 % Ad 3. The call of the macro |tip| has the form |tip|(pen,
838 % pre_elongate, post_elongate)(list_of_nodes), where “pen”
839 % and “list_of_nodes” have the same meaning as previously.
840 % In particular, a pen can be omitted. At corner nodes
841 % specified in the list of nodes, the consecutive elements of the outline
842 % are not joined with an appropriate subpath of a pen; instead, they
843 % are elongated (extrapolated) until they intersect. This process corresponds
844 % (roughly) to the \MP{ setting |linejoin:=mitered|:
845
846 % The illustration above is the result of the following call
847 % of the macro |pen_stroke| (the macro |tip| is invoked with default
848 % settings, only the number of a node is specified):
849 % |pen_stroke(tip()(3))(p)(q); draw q;|
850 % The optional parameters |pre_elongation| and |post_elongation| define how
851 % far the consecutive edges (segments) should be elongated in order to make

```

```

852 % them intersect each other (the measure is the time). If one parameter is
853 % omitted, both will receive the same value; if both are omitted, a default
854 % value, |(0.5,0.5)| (it corresponds to elongation by circa 50\%), will be
855 % used. The precise meaning of the pre- and post-elongation is defined as
856 % follows: for a given pre-edge |e1|, post-edge |e2|, pre-elongation |v1|
857 % and post-elongation |v2|, the paths
858 % |extrapolate (0, 1/(1+v1) of e1| and
859 % |extrapolate (v2/(1+v2), 1) of e2| are computed
860 % (i.e., for the default elongation: |extrapolate (0, 2/3) of s1|
861 % and |extrapolate (1/3, 1) of s2|, respectively).
862 % If elongated curves do not intersect, the terminal nodes
863 % of the consecutive segments are joined with a straight line. The latter
864 % property can be used to obtain a result corresponding to the \MP{} setting
865 % |linejoin:=beveled|: it suffices to apply a null elongation, i.e.,
866 % |tip|(0)(list_of_nodes). Changing the first (empty) parameter
867 % of the |tip| macro in the previous example would yield the following
868 % result:
869
870 % Ad 4. The macro |ignore_directions| has a different character. It is
871 % invoked with one parameter being a comma-separated list of nodes:
872 % |ignore_directions|(list_of_nodes). The numbers {\it must be\} followed
873 % by suffixes |ll| or |rl|. The macro causes that, at specified nodes,
874 % the direction of the outline is not forced to be parallel to the direction
875 % of the path |p| (which is the default); instead, the direction is
876 % calculated by \MP{}. Suffixes determine whether the direction
877 % is not to be forced at the right (|rl|) or the left (|ll|) edge (with
878 % respect to the direction of the path |p|). This heuristic
879 % trick can be used to improve the appearance of the outline
880 % if the "inner" part of the envelope has too tight arcs.
881 %% The examples of the usage of this macro can be found in the \MP{} version
882 %% of D. E. Knuth's 'logo' font (letters 'P' and 'S').
883
884 vardef pen_stroke(text opts)(expr p)(suffix result) =
885   forsuffices $=,r,l,b,e:
886     if not path result$: scantokens("path " & genericize(str result$)); fi
887   endfor
888   save a_, a__, d_, i_, k_, n_, p_, z_, norm_, norml_, normr_, normlr_,
889     fix_opts_, ignore_dir_, ignore_dir__, local_nib_, local_nib__,
890     local_tip_, default_tip_, local_tip__, % internal
891     all, rel, last, nib, cut, tip, ignore_directions, current_node; % exported
892   numeric ignore_dir__[\\]; pair default_tip_, local_tip__[\\];
893   path local_nib__[\\];
894   pair a_, d_, z__[\\]; path p_;
895   %% xpart norm_ norml_ normr_ normlr_
896   vardef norm_ primary n =
897     if cycle p: n mod last else: if n<0: 0 elseif n>last: last else: n fi fi
898   enddef;
899   vardef norml_ primary n = -norm_ n -1 enddef;

```

```

900 vardef normr_ primary n = norm_ n +1 enddef;
901 vardef normlr_@# primary i=
902   if str @#="l": -norm_(last-i)-1 else: i+1 fi
903 enddef;
904 last=length(p);
905 vardef rel primary a =
906   angle((gendir current_node of p) slant_stroke)+a
907 enddef;
908 def all =
909   hide(% locally we use the prefix rather than postfix notation;
910         % a trick due to the |suffix| parameter of the |allcont_| macro
911   vardef l primary n = (norml_ n,0) enddef;
912   vardef r primary n = (normr_ n,0) enddef) allcont_
913 enddef;
914 def allcont_ suffix $ =
915   $0 for i_:=1 upto last if cycle p: -1 fi: , $i_ endfor
916 enddef;
917 vardef fixopts_(suffix optname)(text nodes) text val =
918 %% intersectiontimes lcont_ rcont_
919   save l, r, lcont_, rcont_;
920   def l = lcont_ whatever enddef;
921   primarydef a lcont_ b = (norml_ a,0) enddef;
922   def r = rcont_ whatever enddef;
923   primarydef a rcont_ b = (normr_ a,0) enddef;
924   for n_:=nodes:
925     if numeric n_:
926       current_node:=norm_ n_;
927       optname[norml_ n_] := optname[normr_ n_]
928     else:
929       current_node:=abs(xpart n_)-1; % inverse of both |norml_| and |normr_|
930       optname[xpart(n_)]
931     fi :=val; % |val| may depend on |current_node|
932   endfor
933 enddef;
934 def nib(text nib_)(text nodes) = % nib and node list
935   fixopts_(local_nib_)(nodes)
936   begingroup
937     p_:=default_nib; for k_:=nib_: p_:=k_; endfor \ p_
938   endgroup;
939 enddef;
940 def cut(text nib_and_ang)(text nodes) = % angle, nib and node list
941   fixopts_(local_nib_)(nodes)
942   begingroup
943     p_:=default_nib;
944     for k_:=nib_and_ang:
945       if numeric k_: a_:=dir(unslant_angle(k_)); else: p_:=k_; fi
946     endfor
947     d_:=gendir current_node of p;

```

```

948 z_1:=whatever*a_=tangent_point(d_,p_)+whatever*d_;
949 z_2:=whatever*a_=tangent_point(-d_,p_)+whatever*d_;
950 z_1-z_2
951 endgroup;
952 enddef;
953 def tip(text nib_and_lim)(text nodes)= % limit(s) and node list
954 i_:=0; for n_:=nib_and_lim: if numeric n_: i_[incr i_]:=n_; fi endfor
955 fixopts_(local_tip__)(nodes)
956 elongation_to_times(if i_=0: default_elongation, default_elongation
957 elseif i_=1: i_1, i_1 else: i_1, i_2 fi);
958 fixopts_(local_nib__)(nodes)
959 begingroup
960 p_:=default_nib; for k_:=nib_and_lim: if path k_: p_:=k_; fi endfor \ p_
961 endgroup;
962 enddef;
963 def ignore_directions(text nodes) = % node list
964 fixopts_(ignore_dir__)(nodes) 1;
965 enddef;
966 if default_cap=0:
967 if not cycle p: cut(rel 90)(0,last); fi
968 elseif default_cap=1: % do nothing
969 else:
970 errhelp "Admissible values are 0, 1; continue, I'll use the value 1!";
971 errmessage "PX: improper 'default_cap' value ("&decimal(default_cap)&")";
972 fi
973 opts;
974
975 if default_join=0:
976 default_tip_:=elongation_to_times(default_elongation, default_elongation);
977 elseif default_join=1: % no tip setting, do nothing
978 elseif default_join=2:
979 default_tip_:= (1,0); % |(1,0)=elongation_to_times(0,0)|
980 else:
981 errhelp "Admissible values are 0, 1, 2; continue, I'll use the value 1!";
982 errmessage "PX: improper 'default_join' value ("&decimal(default_join)&")";
983 fi
984 vardef ignore_dir_@#(expr i) = known ignore_dir__[normlr_@# i] enddef;
985 vardef local_tip_@#(expr i) = if known local_tip__[normlr_@# i]:
986 local_tip__[normlr_@# i] else: default_tip_ fi enddef;
987 vardef local_nib_@#(expr i) = if known local_nib__[normlr_@# i]:
988 local_nib__[normlr_@# i] else: default_nib fi enddef;
989 result.r:=pen_stroke_edge.r(p);
990 result.l:=pen_stroke_edge.l(reverse p);
991 if not cycle p:
992 result.b:=pen_cap(predir infinity of result.l,postdir 0 of result.r,
993 -postdir 0 of p,point 0 of p,local_nib_.l(last),local_nib_.r(0));
994 result.e:=pen_cap(predir infinity of result.r,postdir 0 of result.l,
995 predir last of p,point last of p,local_nib_.r(last), local_nib_.l(0));

```

```

996 result:=result.r && result.e && result.l && result.b && cycle;
997 fi
998 enddef;
999
1000 vardef pen_cap(expr a,b,c,p,niba,nibb)=
1001   if path_eq(niba,nibb): pen_join(a,b,p,niba)
1002   else: pen_join(a,c rotated 90,p,niba)-pen_join(c rotated 90,b,p,nibb)
1003 fi
1004 enddef;
1005
1006 _____
1007

```

Postscript Font Generation

```

1008 % POSTSCRIPT FONT GENERATION
1009
1010 % Note that this has been stripped down a lot from the METATYPE1
1011 % original code; most of the stuff for hinting, ligature tables,
1012 % METAFONT-style proof generation, and so on has been removed
1013 % because it's irrelevant to Tsukurimashou.
1014
1015 vardef pfi_file = jobname & ".pfi" enddef;
1016 vardef pic_file = "piclist" enddef;
1017 vardef dim_file = jobname & ".dim" enddef;
1018
1019 errorstopmode; warningcheck:=1;
1020 % constants for introducing
1021 ignore:=whatever; process:=0; utilize:=1; store:=2;
1022 let semicolon_ = ; % stores original meaning of a semicolon
1023 % if |tracingdimens>0| then |dim_file| is generated
1024 newinternal tracingdimens;
1025
1026 def write_special = % additional info to be processed by AWK
1027   special "%GLYNFO: " &
1028 enddef;
1029 vardef mtone_glyph_pfx = "MT1: glyph " & str glyph_name & ": " enddef;
1030 def mtone_message = message mtone_glyph_pfx & enddef;
1031
1032 % Macro write_tex provides contact with the
1033 % outer world. The macro contains the information about EPSes that is
1034 % used for proofing and assembling the font; must be consistent with
1035 % the definitions contained in the files 'mpform.sty' and 'mp2pf.awk'.
1036 vardef write_tex(expr name, num) =
1037   write "\EPSNAMEandNUMBER{" & name & "}{" & decimal(num) & "}"
1038   to pic_file & ".tex"
1039 enddef;
1040

```

```

1041 % The following macros are related to the operation of slanting.
1042 % In particular, they enable to keep a fixed width of a stem
1043 % after slanting.
1044 vardef slant_ang = % should be rather called "local_slant_angle"
1045 slang \\\ if known glyph_slanting.glyph_name: * glyph_slanting.glyph_name fi
1046 enddef;
1047 vardef slant_val = tand(slant_ang) enddef;
1048 vardef slant_preadjust(expr slope, slang) =
1049 % |if sind(angle(slope))=0: 1 else:|
1050 % |abs(sind(angle(slope))/sind(angle(cotd(angle(slope))+tand(slang),1)))|
1051 % |fi|
1052 % Correction of stem size taking into account its slope and a slant angle;
1053 % nice formula, isn't it? Much simpler than the previous one, yet equivalent:
1054 length(unitvector(slope) slanted tand(slang))
1055 enddef;
1056 vardef slant_stroke_val = slant_val enddef; % compatibility with plain_ex.mp
1057
1058 vardef stem_corr (expr slope) = slant_preadjust(slope, slant_ang) enddef;
1059
1060 def italicized = % fairly complex operation
1061 if slang<>0:
1062   if known glyph_slanting.glyph_name:
1063     if glyph_slanting.glyph_name=0: shifted (math_axis*tand(slang),0) fi
1064   fi
1065   shifted (italic_shift*tand(slang),0) % re-positioning
1066   slanted slant_val % and slanting
1067 fi
1068 enddef;
1069
1070 primarydef b || c =
1071 whatever*b + c*stem_corr(b)*unitvector(b rotated 90)
1072 enddef;
1073
1074 primarydef c /\ b =
1075 % A variant of the ||leg|| procedure that iteratively counteracts slant
1076 % deformation; as with ||leg||, given: |c| – hypotenuse (vector) of
1077 % a right-angled triangle, |b| – the length of one of its legs;
1078 % result: the other leg of the triangle (vector),
1079 if slant_ang=0: (c leg b)
1080 else:
1081   begingroup save b_, b__, n_; b_:=b__:=b; n_:=10;
1082   forever:
1083     b_:=b*stem_corr(c leg b_);
1084     exitif (abs(b_-b__)<.01) or (n_<=0);
1085     b__:=b_; n_:=n_-1;
1086   endfor
1087   if (abs(b_-b__)>=.01):
1088     errhelp "The result is likely to be weird";

```

```

1089   errmessage mtone_glyph_pfx & "iteration hasn't converged";
1090   fi
1091   c leg b_
1092 endgroup
1093 fi
1094 enddef;
1095
1096 % Obsolete?
1097 vardef rib(expr t,p,r) text u = % |u| is either empty or a vector
1098   save k_; pair k_; for i_:=u: k_:=u; endfor
1099   if unknown k_: k_=((udir t of p) rotated 90); fi
1100   (point t of p) + r * k_ * stem_corr(k_ rotated 90)
1101 enddef;
1102
1103 % The operation {\it compose_path\} is useful in \MP{} programs
1104 % automatically generated from PFB sources (pf2mtl utility). Suffixes
1105 % $a$ and $b$ of control nodes stand for 'after' and 'before', respectively;
1106 % The operation {\it compose_path\} makes use of the operation
1107 % {\it compose_segment\} that serves for constructing non-cyclic
1108 % paths. Undefined nodes are ignored.
1109 vardef compose_segment@#(expr m,n) = % |m|<=|n|, not checked
1110   if unknown inside_compose_path_: save idx_, n_; n_:=1; fi
1111   save n__; n__:=n_+1;
1112   for i_:=m upto n: if known @#[i_]: idx_[incr(n_)]:=i_; fi endfor
1113   for i_:=n__ upto n_1:
1114     @#[idx_[i_]] .. controls
1115       @#[idx_[i_]] if known @#[idx_[i_]]a: a fi
1116       and @#[idx_[i_+1]] if known @#[idx_[i_+1]]b: b fi
1117     ..
1118   endfor
1119   @#[idx_[n_]]
1120 enddef;
1121 vardef compose_path@#(expr n) =
1122   save inside_compose_path_, idx_, n_; n_:=1; inside_compose_path_:=1;
1123   compose_segment@#(0,n)
1124   if @#[idx_[0]]=@#[idx_[n_]]: & else: - fi \ cycle
1125 enddef;
1126
1127 % Basic macros for building character glyphs:
1128 vardef round_node_values(expr p) =
1129   save d_; % candidates for Flex - no checking for "straightlinesss"
1130   for t_:=0 upto length(p)-1:
1131     if round(point t_ of p)=round(point t_+1 of p):
1132       hide(mtone_message "degenerated bezier " & ", length=" &
1133         decimal(length(p)) & " " & ", time=" & decimal(t_) & " ";
1134       show p)
1135     else:
1136       round(point t_ of p)..

```



```

1137   if if known d__[t_] or known d__[t_+1]: false else:
1138     is_line(subpath (t_,t_+1) of p) fi:
1139     controls round(point t_ of p) and round(point t_+1 of p)
1140   else:
1141     controls round(postcontrol t_ of p) and round(precontrol t_+1 of p)
1142   fi
1143   ..
1144 fi
1145 endfor
1146 round(point length(p) of p) \ \ if cycle p: & cycle fi
1147 enddef;
1148
1149 primarydef a start b =
1150 if cycle a:
1151   if b=default: default_start_(a)
1152   else: ((subpath (b,length(a)+b) of a) & cycle) fi
1153 else: a fi
1154 enddef;
1155
1156 newinternal default; default:=infinity;
1157 vardef default_start_(expr p) =
1158   save i_,j_,pi_,pj_; pair pi_,pj_;
1159   j_:=0; pj_:=point j_ of p;
1160   for i_=1 upto length(p):
1161     pi_:=point i_ of p;
1162     if (xpart(pi_)>xpart(pj_)) or
1163       (xpart(pi_)=xpart(pj_)) and (ypart(pi_)<ypart(pj_)):
1164       j_:=i_; pj_:=point j_ of p;
1165     fi
1166   endfor
1167   (subpath (j_, length(p)+j_) of p) & cycle
1168 enddef;
1169
1170 def Fill text glist =
1171   begingroup
1172   save h_; path h_;
1173   for g_:=glist:
1174     h_:=g_ start.default; % JMN's suggestion
1175     if turningnumber h_<>1:
1176       errhelp "The result is likely to be weird.";
1177       errmessage mtone_glyph_pfx & "strange turning number in Fill, " &
1178         decimal(turningnumber h_);
1179     fi
1180     if glyph_usage div store = 1: % storing
1181       glyph_stored.glyph_name[incr glyph_stored.glyph_name.num]=h_;
1182     fi
1183     glyph_list[incr glyph_list.num]:=round_node_values(h_ italicized);
1184     update_glyph_bb(glyph_list[glyph_list.num]);

```

```

1185   endfor;
1186 endgroup
1187 enddef;
1188
1189 def unFill text glist =
1190   begingroup
1191   save h_; path h_;
1192   for g_:=glist:
1193     h_:=g_ start.default; % JMN's suggestion
1194     if turningnumber h_<>-1:
1195       errhelp "The result is likely to be weird.";
1196       errmessage mtone_glyph_pfx & "strange turning number in unFill, " &
1197         decimal(turningnumber h_);
1198     fi
1199     if glyph_usage div store = 1: % storing
1200       glyph_stored.glyph_name[incr glyph_stored.glyph_name.num]=h_;
1201     fi
1202     glyph_list[incr glyph_list.num]:=round_node_values(h_ italicized);
1203   endfor;
1204 endgroup
1205 enddef;
1206
1207 def fix_hsbw (expr xr,ml,mr) =
1208   glyph_shift:=round(ml); % shift = left margin
1209   glyph_width:=round(xr+ml+mr); % declared width plus margins
1210   if glyph_usage div store = 1: % storing
1211     glyph_shift.glyph_name:=glyph_shift; glyph_width.glyph_name:=glyph_width;
1212   fi
1213 enddef;
1214
1215 def fix_exact_hsbw(expr xr,ml,mr) =
1216   glyph_shift:=round(ml); % shift = left margin
1217   glyph_width:=xr+ml+mr; % declared width plus margins
1218   if glyph_usage div store = 1: % storing
1219     glyph_shift.glyph_name:=glyph_shift; glyph_width.glyph_name:=glyph_width;
1220   fi
1221 enddef;
1222
1223 % Macros below set PostScript and \TeX{} units; a trick with '\#'
1224 % in {\it tfm\_units\}/} proves useful in achieving compatibility
1225 % with the Knuthian fonts (e.g., it is employed in {\it logo\}/} font).
1226 % Old versions of {\it tfm\_units\}/} and {\it ps\_units\}/} are less
1227 % accurate, but are kept because of backward compatibility reasons.
1228 vardef tfm_units(text x) =
1229   save #; if known (x#): x# else: x/(1000/designsize) fi
1230 enddef;
1231 vardef old_tfm_units(text x) =
1232   save #; if known (x#): x# else: x/1000*designsize fi

```

```

1233 endif;
1234
1235 vardef ps_units(expr x) = x*(1000/designsize) endif;
1236 vardef old_ps_units(expr x) = x/designsize*1000 endif;
1237
1238 def define_ps_units(text t) =
1239   forsuffices $:=t: $:=ps_units($, #); endfor
1240 endif;
1241 def define_whole_ps_units(text t) =
1242   forsuffices $:=t: $:=round(ps_units($, #)); endfor
1243 endif;
1244 def define_even_ps_units(text t) =
1245   forsuffices $:=t: $:=2round(1/2ps_units($, #)); endfor
1246 endif;
1247
1248 % In general, all objects are supposed to be drawn by the
1249 % {\bf endglyph} macro, i.e., all drawing operations are deferred.
1250 % The same concerns labelling, which necessitates redefinition
1251 % of labelling macros.
1252
1253 def label_(suffix pos)(expr s,z, dot_or_not) =
1254 % should be more complex if overlapping labels are to be avoided
1255 endif;
1256 string label_defaultfont; label_defaultfont:="cmr10";
1257 newinternal label_defaultscales; label_defaultscales:=magstep 5;
1258
1259 % If the {\it project\} variable is assigned value greater than 0,
1260 % proofing mode is assumed; the following macros display then
1261 % the details of the construction of glyphs for proofing purposes.
1262 % The larger value of the variable {\it project}, the more details
1263 % are visualised.
1264 def local_drawoptions (text t) = % to be used within a group, see below
1265 % \begingroup \def\#1{\it#1}% local: no underscore hacks
1266   save _op_; drawoptions(t);
1267 % \endgroup
1268 endif;
1269
1270 def update_glyph_bb(expr p) =
1271   if unknown glyph_llx:
1272     glyph_llx:=xpart(llcorner(p)); glyph_lly:=ypart(llcorner(p));
1273     glyph_urx:=xpart(urcorner(p)); glyph_ury:=ypart(urcorner(p));
1274   else:
1275     if xpart(llcorner(p))<glyph_llx: glyph_llx:=xpart(llcorner(p)); fi
1276     if ypart(llcorner(p))<glyph_lly: glyph_lly:=ypart(llcorner(p)); fi
1277     if xpart(urcorner(p))>glyph_urx: glyph_urx:=xpart(urcorner(p)); fi
1278     if ypart(urcorner(p))>glyph_ury: glyph_ury:=ypart(urcorner(p)); fi
1279   fi
1280 endif;

```

```

1281 string stencil_dir;
1282 def ship_glyphs =
1283   begingroup
1284     local_drawoptions();
1285     for g_=1 upto glyph_list.num:
1286       if turningnumber glyph_list[g_] > 0: fill else: unfill fi
1287       glyph_list[g_] shifted (glyph_shift, 0);
1288     endfor
1289   endgroup
1290 enddef;
1291 newinternal show_stroke_size; show_stroke_size:=1.5;
1292 color show_stroke_color; show_stroke_color:=red;
1293
1294 color label_dot_color, label_text_color;
1295 label_dot_color:=.8white; label_text_color:=black;
1296 newinternal label_dot_size; label_dot_size:=3bp;
1297
1298 % Begin and end of the definitions of a character glyph:
1299 def begin_skip =
1300   let endglyph = fi;
1301   let ; = end_skip semicolon__
1302   if false:
1303   enddef;
1304 def end_skip =
1305   let ; = semicolon__ semicolon__
1306   let endglyph = endglyph__;
1307 enddef;
1308
1309 def uni_name(text name) = % name is either a suffix or a string expression
1310   if is_suffix(name):
1311     name
1312   else:
1313     scantokens (begingroup
1314       save rval;
1315       string rval;
1316       rval:="" for i=1 upto length (name):
1317         & "_" & (substring (i-1,i) of (name))
1318       endfor;
1319       rval
1320     endgroup)
1321   fi
1322 enddef;
1323
1324 def glyph_name_ext = enddef;
1325 def beginglyph(text name) =
1326   %
1327   def original_glyph_name = name enddef;
1328   % to use in |endglyph|

```

```

1329 def glyph_name = uni_name(name) glyph_name_ext enddef;
1330 numeric glyph_usage; glyph_usage:=glyph_usage.glyph_name;
1331 if unknown glyph_usage: expandafter begin_skip fi
1332 string ps_name; ps_name:=original_glyph_name;
1333 if unknown ps_name:
1334   errhelp "Use macro 'introduce' or 'assign_name' prior to 'beginglyph.'";
1335   errmessage "MT1: PS name not assigned to " & str glyph_name;
1336 fi
1337 if name_used(original_glyph_name):
1338   errhelp "Proceed if you wish, I'll use the second glyph description.";
1339   errmessage "MT1: double output: name " & (str glyph_name);
1340 fi
1341 if glyph_usage mod store = 1: % utilizing
1342   mark_name_used(original_glyph_name);
1343 fi
1344 numeric glyph_code, glyph_num;
1345 glyph_code:=name_to_code(original_glyph_name);
1346 if glyph_code<0: glyph_num:=500-decr(min_glyph_code); else:
1347   glyph_num:=100+glyph_code;
1348   if glyph_code>max_glyph_code: max_glyph_code:=glyph_code; fi
1349 fi
1350 %
1351 beginfig(glyph_num)
1352 if glyph_usage mod store = 1: % utilizing
1353   write_special "NAME " & ps_name & " " & decimal(glyph_code);
1354   % mpform.sty and mp2pf.awk interface
1355 % |write_tex(glyph_name, glyph_num);|
1356   write_tex(ps_name, glyph_num);
1357 fi;
1358 glyph_list.num:=label_list.num:=0;
1359 path glyph_list[\\];
1360 picture label_list[\\]; pair label_list.dot[\\];
1361 numeric glyph_llx, glyph_lly, glyph_urx, glyph_ury;
1362 numeric bitmap_scale; pair bitmap_offset;
1363 numeric glyph_shift, glyph_width, glyph_axis;
1364 save glyph;
1365 hstem_list.num:=vstem_list.num:=hstem_list.cov:=vstem_list.cov:=0;
1366 pair hstem_list[\\], vstem_list[\\];
1367 path hstem_list_segms[\\], vstem_list_segms[\\];
1368 numeric old_hinting_scheme, new_hinting_scheme;
1369 if glyph_usage div store = 1: % storing
1370   if not path glyph_stored.glyph_name[0]: % glyph_name may contain digits
1371     scantokens("path " & genericize(str glyph_stored.glyph_name) & "[ ]");
1372     scantokens("pair " & genericize(str hstem_stored.glyph_name) & "[ ]");
1373     scantokens
1374       ("path " & genericize(str hstem_stored_segms.glyph_name) & "[ ]");
1375     scantokens("pair " & genericize(str vstem_stored.glyph_name) & "[ ]");
1376     scantokens

```

```

1377     ("path " & genericize(str vstem_stored_segms.glyph_name) & "[]");
1378 fi
1379 glyph_stored.glyph_name.num:=0;
1380 hstem_stored.glyph_name.num:=0; vstem_stored.glyph_name.num:=0;
1381 fi
1382 scantokens extra_beginglyph;
1383 enddef;
1384
1385 picture endglyph_picture;
1386 def endglyph =
1387 scantokens extra_endglyph;
1388 % usually, |currentpicture=nullpicture|, but if not (i.e., some
1389 % extra objects have been drawn), the picture must be shifted:
1390 endglyph_picture:=currentpicture shifted (glyph_shift,0);
1391 currentpicture:=nullpicture;
1392 if known glyph_axis: % actually, used only with stored chars
1393 glyph_axis.glyph_name:=glyph_axis;
1394 fi
1395 % fix char dimensions and write them to TFM and/or |dim_file|
1396 % independently of |glyph_usage| (|dim_file|)
1397 % fix_tfm_data(glyph_urx+glyph_shift, glyph_ury);
1398 if glyph_usage mod store = 1: % utilizing
1399 write_special "HSBW * " & decimal(glyph_width);
1400 write_special "BEGINCHAR";
1401 ship_glyphs;
1402 endfig;
1403 else:
1404 endgroup; % ends figure without shipping it out
1405 fi
1406 enddef;
1407 let endglyph_=endglyph;
1408 string extra_beginglyph, extra_endglyph; extra_beginglyph=extra_endglyph="";
1409
1410 % Additional macros
1411 vardef fix_name_list text t =
1412 string name_list[]; numeric name_list.num; name_list.num:=0;
1413 save , ; let , = fix_name_list_; fix_name_list_ t
1414 enddef;
1415 def fix_name_list__ suffix name =
1416 ; % important semicolon!
1417 if str name<>"": fix_name_list_s__ name else: fix_name_list_e__ "" & fi
1418 enddef;
1419 def fix_name_list_s__ suffix s_name =
1420 fix_name_list_e__ (str s_name)
1421 enddef;
1422 def fix_name_list_e__ expr e_name =
1423 % name is expected to be of the string type
1424 name_list[incr name_list.num]=e_name

```

```

1425 enddef;
1426
1427 def introduce suffix name =
1428   if str name="": introduce_
1429   elseif (substring (0,1) of str name)<>"_": introduce_ name
1430   else: introduce__ name fi
1431 enddef;
1432 def introduce_ expr name = % name is expected to be a string expression
1433   introduce__ uni_name(name)
1434 enddef;
1435 vardef introduce__@#(expr usage, slanting)(text stencil) =
1436   if (unknown process_selected or known process_selected@#)
1437     and known usage and unknown ignore_selected@#:
1438     glyph_usage@#:=usage;
1439     % ignore=whatever|, |process=0|, |utilize=1|, |store=2|
1440     if unknown glyph_ps_name@#: % set default:
1441       assign_name @# (substring (1,infinity) of (str @#));
1442     fi
1443     glyph_slanting@#:=slanting;
1444     % ignore |slant_ang| if |0|; use |slant_ang| otherwise
1445     % |stencil| can be either string (recommended) or suffix (with default
1446     % extension |"eps"| – obsolete), hence some trickery below
1447     save r_; string r_;
1448     for i_:=stencil: if string i_: r_:=i_; fi endfor
1449     if unknown r_:
1450       forsuffices i_:=stencil: r_:= str i_; endfor
1451       if r_<>"": r_:=r_ & "eps"; fi
1452     fi
1453     if r_<>"":
1454       if not string glyph_stencil@#:
1455         scantokens("string " & genericize(str glyph_stencil@#));
1456       fi
1457       glyph_stencil@# = r_;
1458     fi
1459   fi
1460 enddef;
1461
1462 vardef assign_name@#(expr ps_name) =
1463   if not string glyph_ps_name @#:
1464     scantokens("string " & genericize(str glyph_ps_name@#));
1465   fi
1466   glyph_ps_name@#:=ps_name;
1467 enddef;
1468
1469 def standard_introduce(expr name) =
1470   introduce name (utilize+store)(1)();
1471 enddef;
1472

```

```

1473 vardef name_to_code(text name) =
1474   save res_, name_; string name_;
1475   name_:=name; res_=-1;
1476   for i:=0 upto 255: % 1-to-1 coding presumed
1477     if known code_to_name_[i]: if code_to_name_[i]=name_: res_:=i; fi fi
1478     exitif res_>-1;
1479   endfor
1480   res_
1481 enddef;
1482 def encode(text name)(expr glyph_code)=
1483   if (glyph_code<0) or (glyph_code>255):
1484     errhelp "The code must be within the range 0..255";
1485     errmessage "MT1: improper code " & decimal(glyph_code) &
1486       " ('encode' ignored)";
1487   elseif known code_to_name_[glyph_code]:
1488     errhelp "A given code can be assigned only to one name (obviously)";
1489     errmessage "MT1: repeated code for " & code_to_name_[glyph_code] &
1490       " (" & decimal(glyph_code) & "; 'encode' ignored)";
1491   else:
1492     code_to_name_[glyph_code]:=name;
1493   fi
1494 enddef;
1495 string code_to_name_[\\];
1496
1497 vardef name_used(text name) =
1498   save res_, name_; boolean res_; string name_;
1499   name_:=name; res_:=false;
1500   for i:=1 upto max_name_used:
1501     res_:=(name_used_[i]=name_);
1502     exitif res_;
1503   endfor
1504   res_
1505 enddef;
1506 def mark_name_used(text name)=
1507   name_used_[incr max_name_used]:=name;
1508 enddef;
1509 string name_used_[\\]; newinternal max_name_used;
1510
1511 vardef string_date =
1512   if day<10: "0" & fi decimal(day) & " " &
1513   if month<10: "0" & fi decimal(month) & " " &
1514   decimal(year)
1515 enddef;
1516
1517 def set_pfi (suffix kind) (expr val) =
1518   if known val:
1519     if (numeric val) or (string val) or (boolean val):
1520       if (numeric val) and (not numeric pf_info_set.kind):

```



```

1521     scantokens ("numeric " & genericize(str pf_info_set.kind));
1522 elseif (string val) and (not string pf_info_set.kind):
1523     scantokens ("string " & genericize(str pf_info_set.kind));
1524 elseif (boolean val) and (not boolean pf_info_set.kind):
1525     scantokens ("boolean " & genericize(str pf_info_set.kind));
1526 fi
1527 pf_info_set.kind:=val;
1528 write str kind & " : " &
1529     if string val: val
1530     elseif numeric val: decimal(val)
1531     elseif boolean val: if val: "true" else: "false" fi
1532 fi
1533 to pfi_file;
1534 else:
1535     errhelp "Proceed, I'll just ignore the setting";
1536     errmessage "MT1: pf_info keys can only be of numeric, string " &
1537         "and boolean type";
1538 fi
1539 fi
1540 endif;
1541
1542 def pf_info_version expr v = set_pfi(VERSION,v); endif;
1543
1544 def pf_info_creationdate text t =
1545     begingroup
1546     save k_; k_:=0;
1547     for t_:=t: k_:=k_+1; set_pfi(CREATION_DATE, t_); exitif k_=1; endfor
1548     if k_=0: set_pfi(CREATION_DATE, string_date); fi
1549 endgroup
1550 endif;
1551
1552 def pf_info_fontname text t =
1553     begingroup
1554     save k_; k_:=0;
1555     for t_:=t: k_:=k_+1;
1556         if k_=1: set_pfi(FONT_NAME, t_); fi
1557         if k_=2: set_pfi(FULL_NAME, t_); fi
1558     exitif k_=2;
1559 endfor
1560 if k_=1: set_pfi(FULL_NAME, pf_info_set.FONT_NAME); fi
1561 endgroup
1562 endif;
1563
1564 def pf_info_author expr v = set_pfi(AUTHOR,v); endif;
1565 % There is 'much ado about nothing', i.e., about the sign of descender:
1566 % in a PFB file in an 'ADL' comment, descender is positive, while in an AFM
1567 % in a 'Descender' comment – negative; we will distinguish between
1568 % the two, the more so as 'ADL' comment is not mentioned in

```

```

1569 % in the Adobe documentation {\it Adobe Type 1 Font Format}.
1570
1571 def pf_info_ascender expr v = ascender:=v; set_pfi(ASCENDER,v); enddef;
1572 def pf_info_descender expr v = descender:=v; set_pfi(DESCENDER,v); enddef;
1573
1574 def pf_info_adl text t =
1575   begingroup
1576     save k_; k_:=0;
1577     for t_:=t: k_:=k_+1;
1578       if (k_==1) and known t_: adl_ascender:=t_; set_pfi(ADL_ASCENDER,t_); fi
1579       if (k_==2) and known t_: adl_descender:=t_; set_pfi(ADL_DESCENDER,t_); fi
1580       if (k_==3) and known t_: adl_lineskip:=t_; set_pfi(ADL_LINESKIP,t_); fi
1581       exitif k_==3;
1582     endfor
1583   endgroup
1584 enddef;
1585
1586 def pf_info_underline text t =
1587   begingroup
1588     save k_; k_:=0;
1589     for t_:=t: k_:=k_+1;
1590       if k_==1: set_pfi(UNDERLINE_POSITION,t_); fi
1591       if k_==2: set_pfi(UNDERLINE_THICKNESS,t_); fi
1592       exitif k_==2;
1593     endfor
1594   endgroup
1595 enddef;
1596
1597 def pf_info_pfm text t =
1598   % parameters: name, bold (0 or 1), italic (0 or 1), char set;
1599   % each of them can be either known or unknown or "*" (which means unknown);
1600   % the last parameter can be either numeric or string representation of
1601   % a valid Perl numeric value (e.g., "0xFF" means 255).
1602   begingroup
1603     save k_; k_:=0;
1604     for t_:=t: k_:=k_+1;
1605       if (k_==1) and known t_: set_pfi(PFM_NAME,t_); fi
1606       if (k_==2) and known t_: set_pfi(PFM_BOLD,t_); fi
1607       if (k_==3) and known t_: set_pfi(PFM_ITALIC,t_); fi
1608       if (k_==4) and known t_: set_pfi(PFM_CHARSET,t_); fi
1609       exitif k_==4;
1610     endfor
1611   endgroup
1612 enddef;
1613
1614 def pf_info_fixedpitch expr v = set_pfi(FIXED_PITCH,v); enddef;
1615 def pf_info_capheight expr v = uc_height:=v; set_pfi(CAPHEIGHT,v); enddef;
1616 def pf_info_weight expr v = set_pfi(WEIGHT,v); enddef;

```

```

1617 def pf_info_stdvstem expr v = set_pfi(STDVW,v); enddef;
1618 def pf_info_stdhstem expr v = set_pfi(STDHW,v); enddef;
1619 def pf_info_forcebold expr v = set_pfi(FORCE_BOLD,v); enddef;
1620
1621 % TeX-related font info (fontdimens and headerbytes):
1622 def pf_info_fontdimen text t = % exceptionally, TFM units expected
1623   begingroup
1624     save i_, k_, b_; boolean b_;
1625     k_:=0;
1626     if true for t_:=t: hide(k_:=k_+1) and known t_ endfor and (k_<=3):
1627       k_:=0;
1628       for t_:=t: k_:=k_+1;
1629         if k_==1:
1630           i_:=t_;
1631           % |b| means "we are ready to override (possibly) the previous value
1632           % of a font parameter unless we are inside |complete_tfm_info| and
1633           % then we want to set only a 'virgin' value."
1634           b_:=unknown completing_tfm_info or unknown pf_info_set.FONT_DIMEN[i_];
1635         fi
1636         if b_ and (k_==2): set_pfi(FONT_DIMEN[i_],t_); fontdimen i_: t_; fi
1637         if b_ and (k_==3): set_pfi(DIMEN_NAME[i_],t_); fi
1638       endfor
1639       if b_ and (k_==2): set_pfi(DIMEN_NAME[i_],"(unknown fontdimen name)"); fi
1640     else:
1641       errhelp "Proceed, I'll just ignore TFM fontdimen settings.";
1642       errmessage "MT1: invalid TFM fontdimen data";
1643     fi
1644   endgroup
1645 enddef;
1646 def pf_info_headerbyte text t =
1647   begingroup
1648     save i_, k_; k_:=0;
1649     if true for t_:=t: hide(k_:=k_+1) and known t_ endfor and (k_==2):
1650       k_:=0;
1651       for t_:=t: k_:=k_+1;
1652         if k_==1: i_:=t_; fi
1653         if k_==2:
1654           set_pfi(HEADER_BYTE[i_],if numeric t_: decimal(t_) else: t_ fi);
1655           if i_==9: % encoding scheme, e.g., |"TEX TEXT"|
1656             headerbyte 9: BCPL_string(t_,40); fi
1657           if i_==49: % font family, e.g., |"CMR"|
1658             headerbyte 49: BCPL_string(t_,20); fi
1659           if i_==72: % family member number, which should be between 0 and 255
1660             headerbyte 72: t_; fi
1661         fi
1662       endfor
1663     else:
1664       errhelp "Proceed, I'll just ignore TFM headerbyte settings.";

```

```

1665   errmessage "MT1: invalid TFM headerbyte data";
1666 fi
1667 endgroup
1668 enddef;
1669 def pf_info_designsize expr v = % |designsize| is special
1670   designsize:=v; set_pfi(DESIGN_SIZE,decimal(v) & " (in points)");
1671 enddef;
1672 def pf_info_italicangle expr v =
1673   begingroup
1674     save tfm_units; vardef tfm_units(text x) = c enddef;
1675     slang:=v; set_pfi(ITALIC_ANGLE,-v);
1676     pf_info_fontdimen 1, if known slant: slant else: tand(slang) fi, "(slant)";
1677   endgroup
1678 enddef;
1679 def pf_info_space text t = % three in one
1680   begingroup
1681     save k_; k_:=0;
1682     for t_:=t: k_:=k_+1;
1683       if (designsize<>0) and known t_:
1684         if k_:=1:
1685           space:=t_; pf_info_fontdimen 2, tfm_units(space), "(space)";
1686         elseif k_:=2:
1687           space_stretch:=t_; pf_info_fontdimen 3, tfm_units(space_stretch),
1688             "(space stretch)";
1689         elseif k_:=3:
1690           space_shrink:=t_; pf_info_fontdimen 4, tfm_units(space_shrink),
1691             "(space shrink)";
1692         fi
1693       fi
1694     exitif k_:=3;
1695   endfor
1696 endgroup
1697 enddef;
1698 def pf_info_normal_space text t =
1699   begingroup
1700     save k_; k_:=0;
1701     if true for t_:=t: hide(k_:=k_+1) endfor and (k_<=2):
1702       k_:=0;
1703       for t_:=t: k_:=k_+1;
1704         if (k_:=1) and known t_: space:=t_; fi
1705         if (k_:=2) and known t_: % |t_| is expected to be in TFM units
1706           pf_info_fontdimen 2, t_, "(space)";
1707         fi
1708       endfor
1709       if (k_:=1) and (designsize<>0) and known space:
1710         pf_info_fontdimen 2, tfm_units(space), "(space)";
1711       fi
1712     fi

```

```

1713 endgroup
1714 enddef;
1715 def pf_info_space_stretch text t =
1716 begingroup
1717 save k_; k_:=0;
1718 if true for t_:=t: hide(k_:=k_+1) endfor and (k_<=2):
1719 k_:=0;
1720 for t_:=t: k_:=k_+1;
1721 if (k_==1) and known t_: space_stretch:=t_; fi
1722 if (k_==2) and known t_: % |t_| is expected to be in TFM units
1723 pf_info_fontdimen 3, t_, "(space stretch)";
1724 fi
1725 endfor
1726 if (k_==1) and (designsize<>0) and known space_stretch:
1727 pf_info_fontdimen 3, tfm_units(space_stretch), "(space stretch)";
1728 fi
1729 fi
1730 endgroup
1731 enddef;
1732 def pf_info_space_shrink text t =
1733 begingroup
1734 save k_; k_:=0;
1735 if true for t_:=t: hide(k_:=k_+1) endfor and (k_<=2):
1736 k_:=0;
1737 for t_:=t: k_:=k_+1;
1738 if (k_==1) and known t_: space_shrink:=t_; fi
1739 if (k_==2) and known t_: % |t_| is expected to be in TFM units
1740 pf_info_fontdimen 4, t_, "(space shrink)";
1741 fi
1742 endfor
1743 if (k_==1) and (designsize<>0) and known space_shrink:
1744 pf_info_fontdimen 4, tfm_units(space_shrink), "(space shrink)";
1745 fi
1746 fi
1747 endgroup
1748 enddef;
1749 def pf_info_xheight text t =
1750 begingroup
1751 save k_; k_:=0;
1752 if true for t_:=t: hide(k_:=k_+1) endfor and (k_<=2):
1753 k_:=0;
1754 for t_:=t: k_:=k_+1;
1755 if (k_==1) and known t_: lc_height:=t_; set_pfi(XHEIGHT, t_); fi
1756 if (k_==2) and known t_: % |t_| is expected to be in TFM units
1757 pf_info_fontdimen 5, t_, "(xheight)";
1758 fi
1759 endfor
1760 if (k_==1) and (designsize<>0) and known lc_height:

```

```

1761   pf_info_fontdimen 5, tfm_units(lc_height), "(xheight)";
1762   fi
1763   fi
1764 endgroup
1765 enddef;
1766 def pf_info_quad text t =
1767   begingroup
1768   save k_; k_:=0;
1769   if true for t_:=t: hide(k_:=k_+1) endfor and (k_<=2):
1770     k_:=0;
1771     for t_:=t: k_:=k_+1;
1772       if (k_==1) and known t_: quad:=t_; fi
1773       if (k_==2) and known t_: % |t_| is expected to be in TFM units
1774         pf_info_fontdimen 6, t_, "(quad)";
1775       fi
1776     endfor
1777     if (k_==1) and (designsize<>0) and known quad:
1778       pf_info_fontdimen 6, tfm_units(quad), "(quad)";
1779     fi
1780   fi
1781 endgroup
1782 enddef;
1783 def pf_info_extra_space text t =
1784   begingroup
1785   save k_; k_:=0;
1786   if true for t_:=t: hide(k_:=k_+1) endfor and (k_<=2):
1787     k_:=0;
1788     for t_:=t: k_:=k_+1;
1789       if (k_==1) and known t_: extra_space:=t_; fi
1790       if (k_==2) and known t_: % |t_| is expected to be in TFM units
1791         pf_info_fontdimen 7, t_, "(extra space)";
1792       fi
1793     endfor
1794     if (k_==1) and (designsize<>0) and known extra_space:
1795       pf_info_fontdimen 7, tfm_units(extra_space), "(extra space)";
1796     fi
1797   fi
1798 endgroup
1799 enddef;
1800 def pf_info_encoding text t =
1801   begingroup
1802   save k_; k_:=0;
1803   for t_:=t: k_:=k_+1;
1804     if (k_==1) and known t_: if t_<>"": set_pfi(ENCODING_SCHEME, t_); fi fi
1805     if (k_==2) and known t_: if t_<>"": pf_info_headerbyte 9, t_; fi fi
1806     if (k_==3) and known t_: if t_<>"": set_pfi(ENCODING_NAME, t_); fi fi
1807     exitif k_=3;
1808   endfor

```

```

1809 if (k_=1) and known pf_info_set.ENCODING_SCHEME % upward compatibility
1810   and unknown pf_info_set.HEADER_BYTE9:
1811   pf_info_headerbyte 9, pf_info_set.ENCODING_SCHEME;
1812   fi
1813 endgroup
1814 endif;
1815 def pf_info_familyname text t =
1816 begingroup
1817   save k_; k_:=0;
1818   for t_:=t: k_:=k_+1;
1819     if k_=1: set_pfi(FAMILY_NAME, t_); fi
1820     if k_=2: pf_info_headerbyte 49, t_; fi
1821     exitif k_=2;
1822   endfor
1823   if k_=1: pf_info_headerbyte 49, pf_info_set.FAMILY_NAME; fi
1824 endgroup
1825 endif;
1826
1827 % bluezz forever...
1828 newinternal blue_fuzz, blue_scale, blue_shift;
1829 blue_fuzz:=0; % Adobe Type 1 Font Format, p. 41
1830 blue_scale:=0.0454545;
1831 blue_shift:=7;
1832
1833 % it is advisable to avoid typso whenever possible:
1834 def show_compose expr x = show_compose_ :=x; enddef;
1835 def show_fills expr x = show_fills_ :=x; enddef;
1836 def show_strokes expr x = show_strokes_ :=x; enddef;
1837 def show_paths expr x = show_paths_ :=x; enddef;
1838 def show_labels expr x = show_labels_ :=x; enddef;
1839 def show_boxes expr x = show_boxes_ :=x; enddef;
1840 def show_stems expr x = show_stems_ :=x; enddef;
1841 def show_stencils expr x = show_stencils_ :=x; enddef;
1842
1843 string extra_beginfont, extra_endfont; extra_beginfont=extra_endfont="";
1844
1845 def beginfont =
1846   min_glyph_code=max_glyph_code=0;
1847   complete_param_setting;
1848   scantokens extra_beginfont;
1849 enddef;
1850
1851 def complete_param_setting =
1852   if designsize=0: designsize:=10; fi
1853   if unknown space: space:=333; fi
1854   if unknown space_stretch: space_stretch:=round(1/2space); fi
1855   if unknown space_shrink: space_shrink:=round(1/3space); fi
1856   if unknown extra_space: extra_space:=round(1/3space); fi

```

```

1857 if unknown quad: quad:=1000; fi
1858 if unknown slang:
1859   if known slant: % compatibility with the Old Tradition...
1860     slang:=angle(1, slant);
1861   else: slang:=0; fi
1862 fi
1863 if unknown uc_height: uc_height:=750; fi
1864 if unknown lc_height: lc_height:=400; fi
1865 if unknown italic_shift: italic_shift:=-40; fi % used to be |-100|
1866 if unknown depth: depth:=-250; fi
1867 if unknown ascender: ascender:=uc_height; fi
1868 if unknown descender: descender:=depth; fi
1869 if unknown adl_ascender: adl_ascender:=uc_height; fi
1870 if unknown adl_descender: adl_descender:=depth; fi
1871 if unknown adl_lineskip: adl_lineskip:=0; fi
1872 if unknown top_line: top_line:=adl_ascender+1/2adl_lineskip; fi
1873 if unknown bot_line: bot_line:=- (adl_descender+1/2adl_lineskip); fi
1874 if unknown math_axis: math_axis:=250; fi
1875 if unknown math_rule: math_rule:=40; fi
1876 begingroup
1877   save rth_, pt_, subs_, desc_depth_, fig_height_, asc_height_;
1878   rth_:=math_rule; pt_:=100;
1879   % math symbol font parameters (defaults excerpted from cmsy10)
1880   subs_:=7/10;
1881   desc_depth_:=70/36pt_; fig_height_:=232/36pt_; asc_height_:=250/36pt_;
1882   if unknown num_one:
1883     num_one:=math_axis+3.51rth_+54/36pt_+subs_*desc_depth_; fi
1884   if unknown num_two: num_two:=math_axis+1.51rth_+30/36pt_; fi
1885   if unknown num_three: num_three:=math_axis+1.51rth_+48/36pt_; fi
1886   if unknown denom_one:
1887     denom_one:=3.51rth_+subs_*fig_height_+124/36pt_-math_axis; fi
1888   if unknown denom_two:
1889     denom_two:=1.51rth_+subs_*fig_height_+30/36pt_-math_axis; fi
1890   if unknown sup_one: sup_one:=899pt_-subs_*asc_height_; fi
1891   if unknown sup_two: sup_two:=849pt_-subs_*asc_height_; fi
1892   if unknown sup_three: sup_three:=104/36pt_; fi
1893   if unknown sub_one: sub_one:=54/36pt_; fi
1894   if unknown sub_two: sub_two:=-849pt_+2subs_*asc_height_+3.1rth_; fi
1895   if unknown sup_drop: sup_drop:=subs_*asc_height_-36/36pt_; fi
1896   if unknown sub_drop: sub_drop:=18/36pt_; fi
1897   if unknown delim_one: delim_one:=239pt_; fi
1898   if unknown delim_two: delim_two:=10.1pt_; fi
1899   % math extension font parameters (defaults excerpted from cmex10)
1900   if unknown big_op_spacing_one: big_op_spacing_one:=40/36pt_; fi;
1901   if unknown big_op_spacing_two: big_op_spacing_two:=60/36pt_; fi;
1902   if unknown big_op_spacing_three: big_op_spacing_three:=72/36pt_; fi;
1903   if unknown big_op_spacing_four: big_op_spacing_four:=216/36pt_; fi;
1904   if unknown big_op_spacing_five: big_op_spacing_five:=36/36pt_; fi;

```



```

1905 endgroup;
1906 enddef;
1907
1908 def endfont =
1909 scantokens extra_endfont;
1910 complete_pf_info;
1911 complete_tfm_info;
1912 scantokens "end";
1913 enddef;
1914
1915 def complete_pf_info =
1916 if unknown pf_info_set.DESIGN_SIZE: pf_info_designsize designsize; fi
1917 if unknown pf_info_set.VERSION: pf_info_version "0.000"; fi
1918 if unknown pf_info_set.AUTHOR: pf_info_author "Unknown"; fi
1919 if unknown pf_info_set.CREATION_DATE: pf_info_creationdate; fi
1920 if unknown pf_info_set.FAMILY_NAME: pf_info_familyname "Untitled"; fi
1921 if unknown pf_info_set.FONT_NAME: pf_info_fontname "Untitled"; fi
1922 if unknown pf_info_set.ASCENDER: pf_info_ascender ascender; fi
1923 if unknown pf_info_set.DESCENDER: pf_info_descender descender; fi
1924 if unknown pf_info_set.ADL_ASCENDER:
1925   pf_info_adl adl_ascender, whatever, whatever;
1926 fi
1927 if unknown pf_info_set.ADL_DESCENDER:
1928   pf_info_adl whatever, adl_descender, whatever;
1929 fi
1930 if unknown pf_info_set.ADL_LINESKIP:
1931   pf_info_adl whatever, whatever, adl_lineskip;
1932 fi
1933 if unknown pf_info_set.UNDERLINE_POSITION:
1934   pf_info_underline -200, whatever;
1935 fi
1936 if unknown pf_info_set.UNDERLINE_THICKNESS:
1937   pf_info_underline whatever, math_rule;
1938 fi
1939 if unknown pf_info_set.ITALIC_ANGLE: pf_info_italicangle slang; fi
1940 if unknown pf_info_set.FIXED_PITCH: pf_info_fixedpitch false; fi
1941 if unknown pf_info_set.CAPHEIGHT: pf_info_capheight uc_height; fi
1942 if unknown pf_info_set.XHEIGHT: pf_info_xheight lc_height; fi
1943 if unknown pf_info_set.WEIGHT: pf_info_weight "Normal"; fi
1944 if unknown pf_info_set.STDVW: fi % just ignore
1945 if unknown pf_info_set.STDHW: fi % just ignore
1946 if unknown pf_info_set.FORCE_BOLD: pf_info_forcebold false; fi
1947 if unknown pf_info_set.ENCODING_SCHEME:
1948   pf_info_encoding "FontSpecific", whatever;
1949 fi
1950 if unknown pf_info_set.HEADER_BYTE9:
1951   pf_info_encoding whatever, "UNSPECIFIED";
1952 fi

```

```

1953 if unknown pf_info_set.BLUE_VALUES: set_pfi(BLUE_VALUES, ""); fi
1954 if unknown pf_info_set.OTHER_BLUES: fi % just ignore
1955 if unknown pf_info_set.BLUE_FUZZ: set_pfi(BLUE_FUZZ, blue_fuzz); fi
1956 if unknown pf_info_set.BLUE_SCALE: set_pfi(BLUE_SCALE, blue_scale); fi
1957 if unknown pf_info_set.BLUE_SHIFT: set_pfi(BLUE_SHIFT, blue_shift); fi
1958 % for those who like smart (implicit) systems:
1959 if unknown no_implicit_spaces:
1960   if not name_used("space"):
1961     if unknown glyph_usage._space: introduce _space (utilize)(0)(); fi;
1962     if (name_to_code("space")<0) and (unknown code_to_name_32):
1963       encode("space") (32);
1964     fi
1965     beginglyph("_space") fix_hsbw(space,0,0); endglyph;
1966   fi
1967   if not name_used("nbspace"):
1968     if unknown glyph_usage._nbspace: introduce _nbspace (utilize)(0)(); fi;
1969     %
1970     beginglyph("_nbspace") fix_hsbw(space,0,0); endglyph; % normal space width
1971   fi
1972 fi
1973 enddef;
1974
1975 def complete_tfm_info =
1976 % complete fontdimen info:
1977 % |designsize| is expected to be known
1978 % |slant| dimen has already been set; |xheight| dimen – not necessarily,
1979 % but |pf_info_set.XHEIGHT| is known:
1980 completing_tfm_info:=1;
1981 pf_info_xheight whatever,
1982   if known lc_height#: lc_height# else: tfm_units(pf_info_set.XHEIGHT) fi;
1983 pf_info_normal_space space if known space#: , space# fi;
1984 pf_info_space_stretch space_stretch
1985   if known space_stretch#: , space_stretch# fi;
1986 pf_info_space_shrink space_shrink
1987   if known space_shrink#: , space_shrink# fi;
1988 pf_info_quad quad if known quad#: , quad# fi;
1989 pf_info_extra_space extra_space if known extra_space#: , extra_space# fi;
1990 font_math_rule math_rule;
1991 font_math_axis math_axis;
1992 % complete header info:
1993 pf_info_headerbyte 72, max(0, 254 - round 2designsize);
1994 completing_tfm_info:=whatever;
1995 enddef;
1996
1997 def BCPL_string(expr s,n)= % string |s| becomes an |n|-byte BCPL string
1998 for l:=if length(s)>=n: n-1 else: length(s) fi: l
1999   for k:=1 upto l: , substring (k-1,k) of s endfor
2000   for k:=l+2 upto n: , 0 endfor endfor

```

```

2001 enddef;
2002
2003 % The Old Tradition...
2004 def font_size expr x = designsize:=x enddef;
2005 def font_slant expr x = fontdimen 1: x enddef;
2006 def font_normal_space expr x = fontdimen 2: x enddef;
2007 def font_normal_stretch expr x = fontdimen 3: x enddef;
2008 def font_normal_shrink expr x = fontdimen 4: x enddef;
2009 def font_x_height expr x = fontdimen 5: x enddef;
2010 def font_quad expr x = fontdimen 6: x enddef;
2011 def font_extra_space expr x = fontdimen 7: x enddef;
2012
2013 % A New Tradition...
2014 def def_font_param (suffix param_name)(expr param_num, param_desc) =
2015   def param_name text x =
2016     begingroup save #; % cf. the definition of |tfm_units|
2017     if (known x#) or ((designsize<>0) and known x):
2018       pf_info_fontdimen param_num, tfm_units(x), "(" & param_desc & ")";
2019     fi
2020   endgroup
2021 enddef;
2022 enddef;
2023
2024 def_font_param (font_math_rule, 8, "math rule");
2025 def_font_param (font_math_axis, 22, "math axis");
2026 % symbol fonts
2027 def_font_param (font_num_one, 8, "num1");
2028 def_font_param (font_num_two, 9, "num2");
2029 def_font_param (font_num_three, 10, "num3");
2030 def_font_param (font_denom_one, 11, "denom1");
2031 def_font_param (font_denom_two, 12, "denom2");
2032 def_font_param (font_sup_one, 13, "sup1");
2033 def_font_param (font_sup_two, 14, "sup2");
2034 def_font_param (font_sup_three, 15, "sup3");
2035 def_font_param (font_sub_one, 16, "sub1");
2036 def_font_param (font_sub_two, 17, "sub2");
2037 def_font_param (font_sup_drop, 18, "sup_drop");
2038 def_font_param (font_sub_drop, 19, "sub_drop");
2039 def_font_param (font_delim_one, 20, "delim1");
2040 def_font_param (font_delim_two, 21, "delim2");
2041 % extension fonts
2042 def_font_param (font_big_op_spacing_one, 9, "big_op_spacing1");
2043 def_font_param (font_big_op_spacing_two, 10, "big_op_spacing2");
2044 def_font_param (font_big_op_spacing_three, 11, "big_op_spacing3");
2045 def_font_param (font_big_op_spacing_four, 12, "big_op_spacing4");
2046 def_font_param (font_big_op_spacing_five, 13, "big_op_spacing5");
2047
2048 endinput

```

obstack.mp

```
1 %
2 % Object stack for Tsukurimashou
3 % Copyright (C) 2011, 2012, 2013 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 bobrush = 1165 enddef;
65 def bokeepshape = 1838 enddef;
66 def boserif = 1746 enddef;
67 def bosize = 1393 enddef;
```

```

68 def botip = 1322 enddef;
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]);

```

OBST

```

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[mysp]:=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_bobrush(expr idx,b) =
283     obstackna.bobrush[find_stroke(idx)]:=b;
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)=
```

```

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     begint Suglyph("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, 2015, 2021 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;
74 transform tsu_xf.cap_upper_accent,tsu_xf.low_centre_accent;
75
76 if is_proportional:
77     tsu_xf.accentedcap=identity;
78 else:
79     xpart tsu_xf.accentedcap=1;
80     xypart tsu_xf.accentedcap=yxpart tsu_xf.accentedcap=0;
81     (500,vmetric(0)) transformed tsu_xf.accentedcap=(500,vmetric(0));
82     (500,vmetric(1)) transformed tsu_xf.accentedcap=(500,vmetric(0.82));
83 fi;
84
85 accent_default[anc_upper]=identity shifted (500,vmetric(0.75));
86 accent_default[anc_grave]=identity
87     shifted (500+0.4*tsu_punct_size,vmetric(0.75));
88 accent_default[anc_acute]=identity
89     shifted (500-0.4*tsu_punct_size,vmetric(0.75));
90 accent_default[anc_wide]=identity xscaled 0.75 shifted (500,vmetric(0.75));
91 accent_default[anc_tilde]=identity xscaled 0.75 shifted (500,vmetric(0.75));
92 accent_default[anc_ring]=identity shifted (500,vmetric(0.93));
93 accent_default[anc_caron_comma]=identity shifted (730,vmetric(0.93));
94 accent_default[anc_lower]=identity shifted (500,vmetric(-0.26));
95 accent_default[anc_lower_connect]=identity shifted (500,vmetric(0));
96 accent_default[anc_centre]=identity
97     scaled ((latin_wide_high_r-latin_wide_low_r)/200) shifted centre_pt;
98
99 % this one is for capitals that have NOT been shrunk
100 tsu_xf.cap_upper_accent=identity shifted (500,vmetric(1.10));
101
102 tsu_xf.low_centre_accent=identity
103     scaled ((latin_wide_xheight_r-latin_wide_low_r)/200)
104     shifted (xpart centre_pt,(latin_wide_xheight_r+latin_wide_low_r)/2);
105
106 vardef tsu_accent.shift_anchors(text c)(expr s) =
107     begingroup;
108     save killflag;
109     boolean killflag;
110     for i:=1 upto max_accent_seen:
111         if unknown accent_has_default[i]:
112             killflag:=true;
113         elseif not accent_has_default[i]:
114             killflag:=true;
115         else:
116             killflag:=false;
117         for j=sp-1 downto 1:
118             if obstacktype[j]=otanchor:

```

```

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

```

```

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

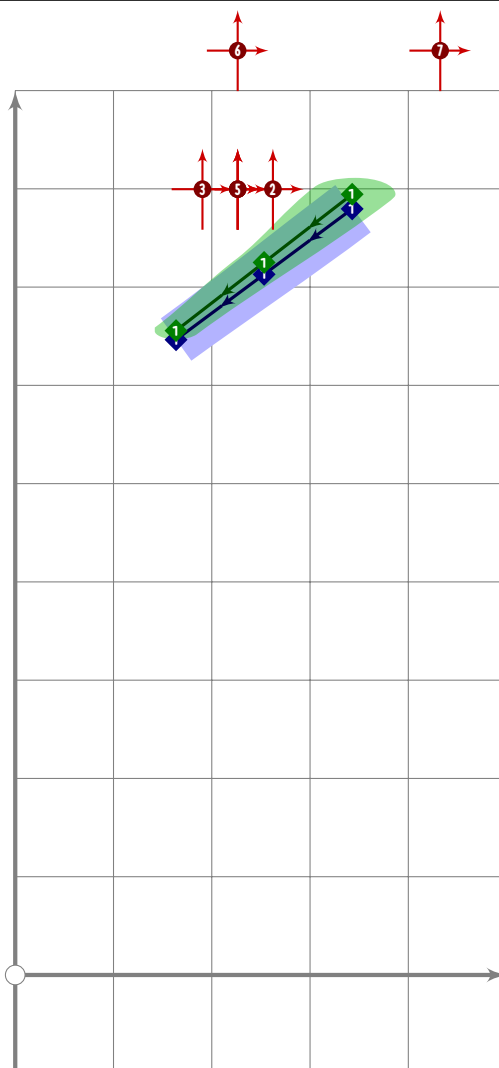
```

accent.mp

```

1 %
2 % Accents for Tsukurimashou
3 % Copyright (C) 2011, 2012, 2013, 2015 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)-(19,19)-(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)-(14,14)-(14,14));
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]);

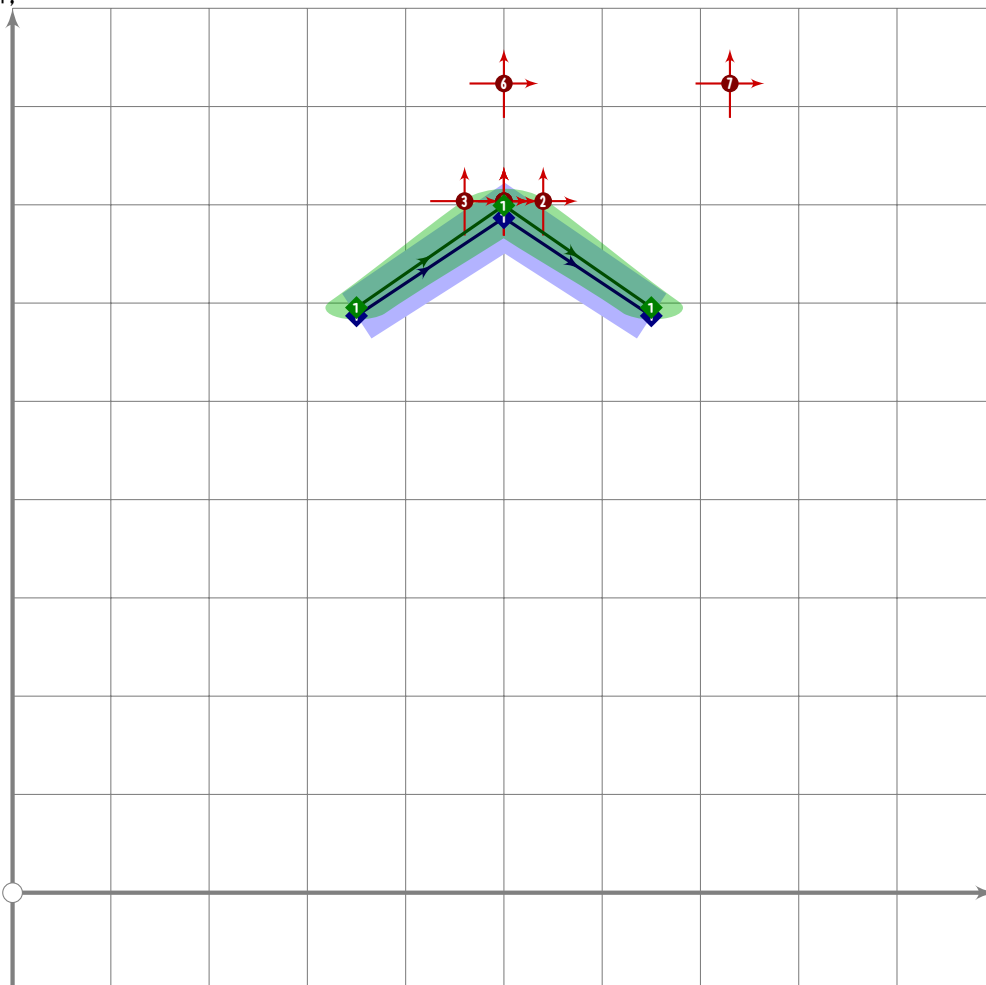
```

U+FF3E
tsuku.uniFF3E

```

90 punct.make_comma((0,0) transformed accent_default[anc_caron_comma],0);
91 endif;
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)-(1.7,1.7)-(1.3,1.3));
99   set_bosize(0,80);
100   set_botip(0,1);
101 endif;

```



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,vmetric(0.80))-
106     (500,vmetric(0.95))-
107     (500+1.5*tsu_punct_size,vmetric(0.80)),
108     (1.6,1.6)-(2,2)-(1.6,1.6));
109   set_bosize(0,80);
110   set_botip(0,1);

```

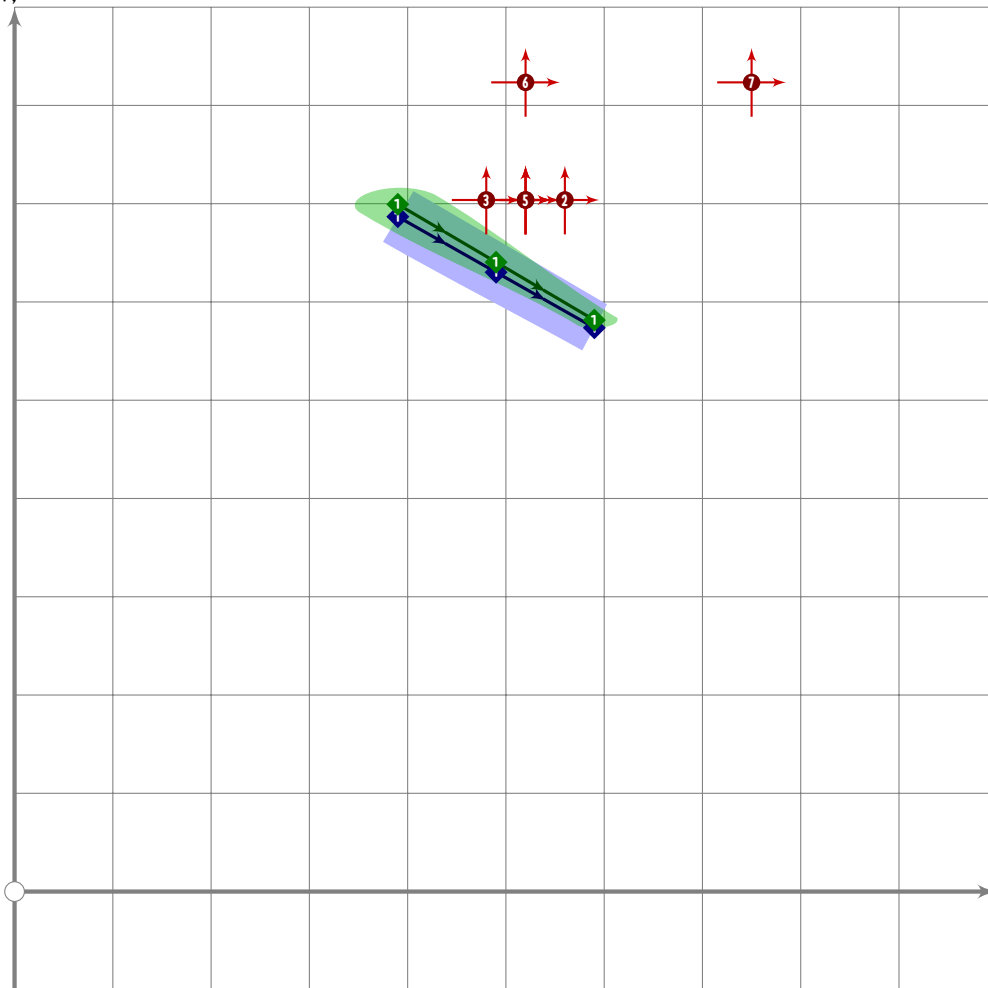
```

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.commbelow =
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);

```

U+FF40
tsuku.uniFF40

159 endif;



ACCE

160

```

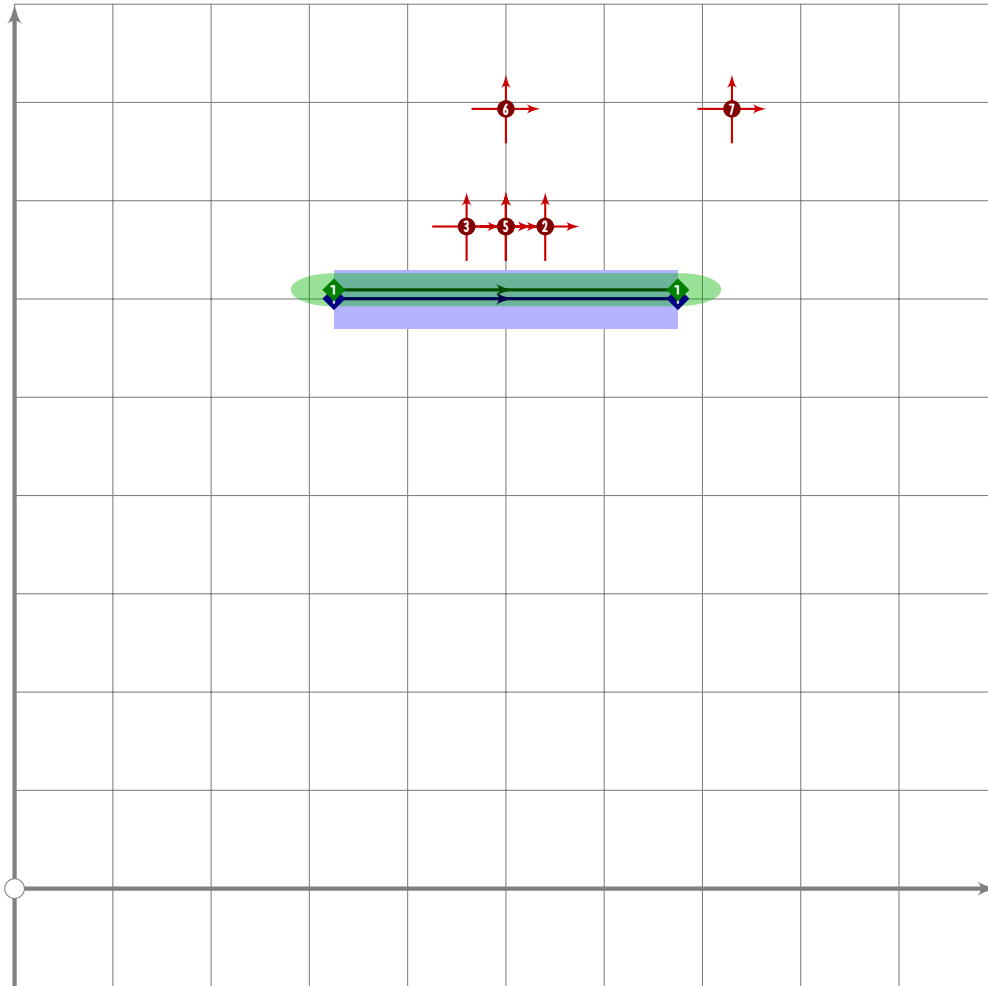
161 vardef tsu_accent.grave =
162   push_anchor(-anc_grave,accent_default[anc_grave]);
163   push_stroke((500-1.1*tsu_punct_size,vmetric(0.95))-
164     (500+0.9*tsu_punct_size,vmetric(0.78)),
165     (2,2)-(1.6,1.6)-(1.3,1.3));
166   replace_strokep(0)(insert_nodes(oldp)(0.5));
167   set_bosize(0,80);
168   set_botip(0,1);
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);
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);
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.brletter)

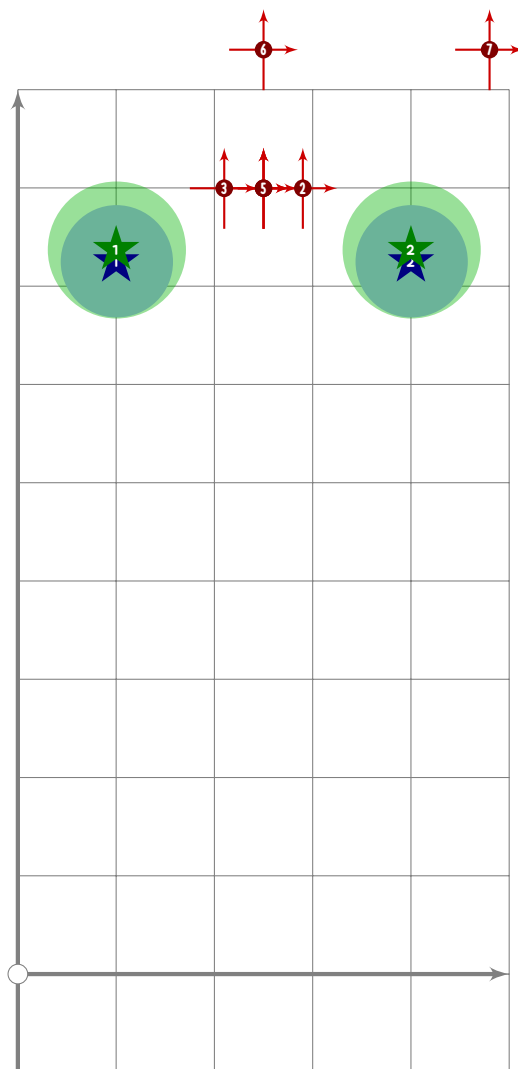
```

```

246     shifted ((500,vmetric(0.93-0.03*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.brletter>10:
251     push_lcblob(reverse fullcircle rotated 45
252         scaled (2*tsu_punct_size-110*tsu_brush_max.brletter)
253         shifted ((500,vmetric(0.93-0.03*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 (0,210));
259 push_anchor(anc_grave,accent_default[anc_grave] shifted (0,210));
260 push_anchor(anc_acute,accent_default[anc_acute] shifted (0,210));
261 push_anchor(anc_wide,accent_default[anc_wide] shifted (0,210));
262 push_anchor(anc_tilde,accent_default[anc_tilde] shifted (0,210));
263 push_anchor(anc_ring,accent_default[anc_ring] shifted (0,210));
264 push_anchor(anc_caron_comma,
265     accent_default[anc_caron_comma] shifted (0,210));
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(oldp)(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;

```

U+00A8
tsuku.dieresis



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_bokeepsape(-1);
306   set_bokeepsape(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=yypart 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=yypart 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;
```

bkencl.mp

```
1 %
2 % Enclosure operations for building kanji
3 % Copyright (C) 2011, 2012, 2013, 2015, 2016, 2021 Matthew Skala
4 %
5-29 [Standard copyright notice]
30
31 inclusion_lock(bkencl);
32
33 

---


34
35 vardef build_kanji.cliff_enclose(text contents) =
36   push_pbox_toexpand("build_kanji.cliff_enclose");
37   perl_structure:=perl_structure&"eids.u2FF8.u5382.1_";
38   push_stroke((50,50)..(120,100)..(160,300)..tension 1.2..(180,760)
39     -(850,760),
40     (1,1)-(1.3,1.3)-(1.5,1.5)-(1.6,1.6)-(1.6,1.6));
41   set_botip(0,3,1);
42   begingroup
43     save t;
44     transform t;
45     (50,50) transformed t=(230,50);
46     (500,50) transformed t=(560,50);
47     (500,850) transformed t=(560,730);
48     tsu_xform(t)(contents);
49   endgroup;
50   expand_pbox;
51 enddef;
52
53 vardef build_kanji.dotcliff_enclose(text contents) =
54   push_pbox_toexpand("build_kanji.dotcliff_enclose");
55   perl_structure:=perl_structure&"eids.u2FF8.u5E7F.2_";
56   push_stroke((550,810)-(550,660),
57     (1.6,1.6)-(1.5,1.5));
58   set_boserif(0,0,10);
59   push_stroke(
60     (50,50)..(120,100)..(160,300)..tension 1.2..(180,660)-(850,660),
61     (1,1)-(1.3,1.3)-(1.5,1.5)-(1.6,1.6)-(1.6,1.6));
62   set_botip(0,3,1);
63   begingroup
64     save t;
65     transform t;
66     (50,50) transformed t=(230,50);
67     (500,50) transformed t=(560,50);
68     (500,850) transformed t=(560,630);
69     tsu_xform(t)(contents);
70   endgroup;
```

```

71 expand_pbox;
72 enddef;
73
74 vardef build_kanji.exist_enclose(text contents) =
75   push_pbox_toexpand("build_kanji.exist_enclose");
76   perl_structure:=perl_structure&"eids.u2FF8.u2E85.1_";
77   build_kanji.sscale(shifted (0,50))(build_kanji.lean_to((380,260)));
78   begingroup
79     save x,y;
80     numeric x[],y[];
81     save t;
82     transform t;
83     z1=(((180,900)-(180,0)) intersectionpoint get_strokep(0))+ (0,-20);
84     x2=x1;
85     y2=-40;
86     push_stroke(z1..z2,(1.6,1.6)-(1.5,1.5));
87     (50,-50) transformed t=(250,-40);
88     (950,-50) transformed t=(910,-40);
89     (950,850) transformed t=(910,640);
90     tsu_xform(t)(contents);
91   endgroup;
92   expand_pbox;
93 enddef;
94
95 vardef build_kanji.flag_enclose(expr xs,ys)(text contents) =
96   push_pbox_toexpand("build_kanji.flag_enclose");
97   perl_structure:=perl_structure&"eids.u2FF8.u5C38.1_";
98   string flag_enclose.ps;
99   flag_enclose.ps:=perl_structure;
100   build_kanji.lr(460,60)
101     (kanji.grtwo.direction;)
102     (perl_structure:=flag_enclose.ps;
103     push_stroke((300,810)..tension 1.2..(220,600)..(110,480),
104       (1.6,1.6)-(1.4,1.4)-(1,1));
105     set_boserif(0,0,10);
106     push_stroke((100,680)-(780,680),(1.6,1.6)-(1.6,1.6));
107     set_boserif(0,1,9);
108     replace_strokep(0)(subpath
109       (0.01+xpart (oldp intersectiontimes get_strokep(-1)),1) of oldp);
110     tsu_xform(identity shifted (-500,0) xyscaled (xs,ys)
111       shifted (500,-20))
112     (contents);
113     flag_enclose.ps:=perl_structure);
114   expand_pbox;
115   perl_structure:=flag_enclose.ps&"_";
116 enddef;
117
118 vardef build_kanji.gate_enclose(text contents) =

```

```

119 perl_structure:=perl_structure&"[build_kanji.gate_enclose";
120 perl_structure:=perl_structure&"eids.u2FF5.u9580.l_";
121 kanji.grtwo.gate;
122 begingroup
123   transform xf;
124   (50,50) transformed xf=(220,40);
125   (950,850) transformed xf=(780,420);
126   xypart xf=yxpart xf=0;
127   tsu_xform(xf)(contents);
128 endgroup;
129 perl_structure:=perl_structure&"[]";
130 endif;
131
132 vardef build_kanji.go_enclose(expr w,o)(text inside) =
133   push_pbox_toexpand("build_kanji.go_enclose");
134   perl_structure:=perl_structure&"eids.u2FF4.u884c.2_1";
135   build_kanji.lcr(480-w/2-20,o)(480+w/2+20,o)
136     (perl_structure:=perl_structure&"[]",[";
137       build_kanji.harmonic(240,0.86,50)(kanji.leftrad.person))
138     (inside)
139     (build_kanji.harmonic(240,0.86,50)(kanji.grthree.thumbtack);
140     perl_structure:=perl_structure&"[");
141   expand_pbox;
142 endif;
143
144 % note this macro can use an "extra_tick" call to draw the tick
145 % seen in some Japanese and Korean characters
146 vardef build_kanji.long_stride_enclose(text contents) =
147   push_pbox_toexpand("build_kanji.long_stride_enclose");
148   perl_structure:=perl_structure&"eids.u2FFA.u5EF4._3";
149   begingroup
150     save myxf;
151     transform myxf;
152     (50,850) transformed myxf=(350,810);
153     (50,50) transformed myxf=(350,80);
154     (950,50) transformed myxf=(950,80);
155     tsu_xform(myxf)(contents);
156     save x,y;
157     numeric x[],y[];
158     x1=80;
159     x2=260;
160     x4=120;
161     y1=y2=780;
162     y4=460;
163     z3=0.4[z4,z2]+mincho*(30,0);
164     z5=(0.3-0.5*mincho)[z4,z2];
165     push_stroke(z1-z2..tension 1.2..z3..z5,
166       (1.6,1.6)-(1.6,1.6)-(1.6-0.1*mincho,1.6-0.1*mincho)-

```



```

167         (1.6-0.2*mincho,1.6-0.2*mincho));
168     set_boserif(0,1,4);
169     set_botip(0,1,0);
170     x7=0;
171     x9=300;
172     x10=220;
173     x11=100;
174     y7=y9=y4;
175     y10=80;
176     y11=-70;
177     z6=point 2.2 of get_stroke(0);
178     z8=(z7-z9) intersectionpoint (get_stroke(0)-((-1)[z4,z2]));
179     push_stroke(z6-z8-z9..tension 1.2..z10..z11,
180         (1.6-0.5*mincho,1.6-0.5*mincho)-(1.5-0.1*mincho,1.5-0.1*mincho)-
181         (1.6,1.6)-(1.4,1.4)-(1,1));
182     set_boserif(0,2,4);
183     set_botip(0,1,1);
184     set_botip(0,2,0);
185     if consume_extra_tick:
186         push_stroke((150,330)..(120,290)..(60,220),
187             (1.2,1.2)-(1,1,1)-(1.6,1.6));
188     fi;
189     push_stroke((130,300)..(390,30)..tension 1.6..(900,50),
190         (1,1)-(1.6,1.6)-(1.9,1.9));
191 endgroup;
192 expand_pbox;
193 enddef;
194
195 vardef build_kanji.road_enclose(text contents) =
196     push_pbox_toexpand("build_kanji.road_enclose");
197     perl_structure:=perl_structure&"eids.u2FFA.u2ECC._3";
198     begingroup
199         save myxf;
200         transform myxf;
201         (50,850) transformed myxf=(315,850);
202         (50,50) transformed myxf=(315,50);
203         (950,50) transformed myxf=(950,50);
204         tsu_xform(myxf)(contents);
205         if consume_extra_tick:
206             push_stroke((130,800)..tension 1.2..(220,720)..(260,660),
207                 (1,1)-(1.3,1.3)-(1.9,1.9));
208             set_bosize(0,92);
209             push_stroke((80,650)..tension 1.2..(160,580)..(200,490),
210                 (1,1)-(1.3,1.3)-(1.9,1.9));
211             set_bosize(0,92);
212             push_stroke((80,383)-(240,380)..(mincho[230,210],200)
213                 ..tension 1.2..(mincho[180,140],50)..{curl 0.4}(60,40),
214                 (1.4,1.4)-(1.6,1.6)-(1.4,1.4)-(1.2,1.2)-(1,1));

```

```

215 else:
216     push_stroke((100,770)..tension 1.2..(180,690)..(220,630),
217         (1,1)-(1.3,1.3)-(1.9,1.9));
218     set_bosize(0,92);
219     push_stroke((80,453)-(240,450)..(mincho[230,210],250)
220         ..tension 1.2..(mincho[180,140],50)..{curl 0.4}(60,-40),
221         (1,1)-(1.6,1.6)-(1,1)-(1.2,1.2)-(1,1));
222 fi;
223 set_botip(0,1,1);
224 set_botip(0,2,1);
225 set_boserif(0,1,4);
226 set_bosize(0,92);
227 push_stroke((point (2.3+mincho) of get_strokep(0))
228     {(1-2*mincho)*direction (2.3+mincho) of get_strokep(0)}..
229     (240,100)..(270,45+15*mincho)..(400,-10)..tension 3..(950,-20),
230     (1,1)-(1.1,1.1)-(1.1,1.1)-(1.7,1.7)-(1.9,1.9));
231 set_bosize(0,92);
232 endgroup;
233 expand_pbox;
234 enddef;
235
236 vardef build_kanji.steam_enclose(expr ur)(text contents) =
237     push_pbox_toexpand("build_kanji.steam_enclose");
238     perl_structure:=perl_structure&"eids.u2FF9,u6C14._4";
239     begingroup
240         save xfa,xfb,xfc,xfd;
241         transform xfa,xfb,xfc,xfd;
242         (50,50) transformed xfc=(50,50);
243         (950,850) transformed xfc=ur;
244         xypart xfc=yxpart xfc=0;
245         tsu_xform(xfc)(contents);
246
247         (0,0) transformed xfa=(0,950) transformed xfc;
248         (1,1) transformed xfa=(280,810);
249         xypart xfa=yxpart xfa=0;
250         (0,0) transformed xfb=(1100,950) transformed xfc;
251         (1,1) transformed xfb=(970,810);
252         xypart xfb=yxpart xfb=0;
253         (0,1) transformed xfd=(1100,950) transformed xfc;
254         (1,0) transformed xfd=(1000,-50);
255         xypart xfd=yxpart xfd=0;
256
257         push_stroke(((1,1)..tension 1.2..(0.5,0.45)..(0,0.2)) transformed xfa,
258             (1.7,1.7)-(1.5,1.5)-(1.2,1.2));
259         set_boserif(0,0,10);
260         set_bosize(0,90);
261
262         push_stroke((get_strokep(0) intersectionpoint

```

```

263      (((0,0.8)-(1,0.8)) transformed xfa))-
264      ((0.5,0.8) transformed xfb),
265      (1.5,1.5)-(1.6,1.6));
266      set__boserif(0,1,9);
267      set__bosize(0,90);
268
269      push__stroke(((0.8,0.4) transformed xfa)-((0.15,0.4) transformed xfb),
270      (1.6,1.6)-(1.6,1.6));
271      set__boserif(0,1,9);
272      set__bosize(0,90);
273
274      push__stroke(((0.2,0) transformed xfa)-(interpath(mincho,
275      (0,1)..tension 1.6..(0.4,0)..(0.6,0)..tension 1.5..
276      (0.73,0.2)..(0.8,0.4),
277      (0,1)..tension 1.6..(0.25,0.2)..{right}(0.8,0){curl 1}..
278      (0.6,0.2)..(0.6,0.4)
279      ) transformed xfd),
280      (1.6,1.6)-(1.6,1.6)-(1.4,1.4)-
281      (1.4,1.4)-(1.2,1.2)-(0.9,0.9));
282      set__boserif(0,1,4);
283      set__botip(0,1,1);
284      set__bosize(0,90);
285      endgroup;
286      expand__pbox;
287  endif;
288
289  vardef build_kanji.wind__enclose(text contents) =
290      push__pbox__toexpand("build_kanji.wind__enclose");
291      push__stroke((50,50)..(120,100)..(160,300)..tension 1.2..(180,760)
292      -interpath(mincho,
293      (770,760){down}{.fdir -72}(810,20)..(860,-10)..tension 1.5..
294      (890,100)..(910,200),
295      (760,760){down}{.fright}(780,100)..{right}(910,-40){curl 1}..
296      (890,60)..(890,160)),
297      (1,1)-(1.3,1.3)-(1.5,1.5)-(1.6,1.6)-(1.6,1.6)-
298      (1.4,1.4)-(1.4,1.4)-(1.2,1.2)-(0.9,0.9));
299      set__botip(0,3,1);
300      set__botip(0,4,1);
301      set__boserif(0,3,4);
302      set__boserif(0,4,4);
303      begingroup
304          save t;
305          transform t;
306          (50,50) transformed t=(230,-50);
307          (950,50) transformed t=(700,-50);
308          (950,850) transformed t=(700,660+20*mincho);
309          tsu__xform(t)(contents);
310      endgroup;

```

```
311 expand__pbox;  
312 endif;
```

BKEN

buildkanji.mp

```
1 %
2 % Build a kanji character by assembling parts
3 % Copyright (C) 2011, 2012, 2013, 2016, 2021 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 endif;
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)-(14,14)-(1.2,1.2));
93 endif;
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 endif;
117
118 vardef build_kanji.attach_fishhook(expr scaleamt)(text curves) =

```

```

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

```

```

263
264 vardef build_kanji.cup(expr ul,lr) =
265   push_stroke(ul-(xpart ul,(-0.2)[ypart lr,ypart ul]),
266     (1.6,1.6)-(1.4,1.4));
267   set_boserif(0,0,10);
268   push_stroke((xpart lr,ypart ul)-(xpart lr,(-0.2)[ypart lr,ypart ul]),
269     (1.6,1.6)-(1.4,1.4));
270   set_boserif(0,0,10);
271   push_stroke((xpart ul,ypart lr)-lr,
272     (1.5,1.5)-(1.5,1.5));
273   perl_structure:=perl_structure&"build_kanji.cup;";
274 enddef;
275
276 vardef build_kanji.harmonic(expr gap,sval,lspread)(text curves) =
277   perl_structure:=perl_structure
278     &"['build_kanji.harmonic',eids.u2FF1u003F._1,['";
279   begingroup
280     save myxf;
281     transform myxf;
282     (50,-50) transformed myxf=(50,-50);
283     (950,-50) transformed myxf=(950,-50);
284     (50,850) transformed myxf=(50,850-gap);
285     tsu_xform(myxf)(curves);
286     perl_structure:=perl_structure&"',[";
287     save hsp;
288     hsp:=sp;
289     tsu_xform(myxf)(build_kanji.spread_legs(lspread)(curves));
290     save i,tp,toclip,nadded;
291     numeric i,toclip,nadded;
292     path tp;
293     i:=0;
294     toclip:=0;
295     nadded:=0;
296     forever:
297       tp:=get_strokep(i);
298       if (ypart ulcorner tp<450)
299         or (abs(ypart (direction 0 of tp)/abs(direction 0 of tp))>0.95)
300         or (abs(ypart (direction infinity of tp)/
301           abs(direction infinity of tp))>0.95):
302         set_bosize(i,0);
303         toclip:=toclip+1;
304       else:
305         replace_strokep(i)
306           (tp shifted -(0.5[ulcorner tp,lrcorner tp])
307             scaled sval
308             shifted ((0,gap)+0.5[ulcorner tp,lrcorner tp]));
309         set_bosize(i)(get_bosize(i)*sqrt(sval));
310         toclip:=toclip+2;

```

```

311         nadded:=nadded+1;
312     fi;
313     i:=i-1;
314     exitif find_stroke(i)<hsp;
315 endfor;
316 endgroup;
317 perl_structure:=perl_structure&"[]";
318 enddef;
319
320 vardef build_kanji.lcr(expr splitpointa,overlapa)(expr splitpointb,overlapb)
321 (text leftstuff)(text centrestuff)(text rightstuff) =
322 perl_structure:=perl_structure
323 &"['build_kanji.lcr';eids.u2ff2._2.1_1.2_';[";
324 begingroup
325     save t;
326     transform t[];
327     yypart t1=yypart t2=yypart t3=1;
328     ypart t1=yxpart t1=xy part t1=0;
329     ypart t2=yxpart t2=xy part t2=0;
330     ypart t3=yxpart t3=xy part t3=0;
331     (50,0) transformed t1=(50,0);
332     (950,0) transformed t1=(splitpointa+overlapa/2,0);
333     (50,0) transformed t2=(splitpointa-overlapa/2,0);
334     (950,0) transformed t2=(splitpointb+overlapb/2,0);
335     (50,0) transformed t3=(splitpointb-overlapb/2,0);
336     (950,0) transformed t3=(950,0);
337     tsu_xform(t1)(leftstuff);
338     perl_structure:=perl_structure&"[]";
339     tsu_xform(t2)(centrestuff);
340     perl_structure:=perl_structure&"[]";
341     tsu_xform(t3)(rightstuff);
342 endgroup;
343 perl_structure:=perl_structure&"[]";
344 enddef;
345
346 vardef build_kanji.lean_to(expr lr) =
347 push_stroke((120,620)-(880,620),(1.6,1.6)-(1.6,1.6));
348 set_boserif(0,19);
349 begingroup
350     save ltxf;
351     transform ltxf;
352     xy part ltxf=yx part ltxf=0;
353     (60,800) transformed ltxf=(60,800);
354     (360,30) transformed ltxf=lr;
355     push_stroke(
356         ((360,800)..tension 1.2..(270,300)..(60,30)) transformed ltxf,
357         (1.6,1.6)-(14,14)-(0,0));
358 endgroup;

```

BUIL

```

359 set__boserif(0,0,10);
360 enddef;
361
362 vardef build_kanji.level(text curves) =
363   begingroup
364     save xsp;
365     xsp:=sp;
366     curves;
367     save lsum,denom,i;
368     lsum:=0;
369     denom:=0;
370     i:=0;
371     forever:
372       exitif find_stroke(i)<xsp;
373       if unknown get__bosize(i):
374         set__bosize(i,100);
375       fi;
376       if (get__bosize(i)>0):
377         lsum:=lsum+mlog(get__bosize(i));
378         denom:=denom+1;
379       fi;
380       i:=i-1;
381     endfor;
382     i:=0;
383     forever:
384       exitif find_stroke(i)<xsp;
385       if get__bosize(i)>0:
386         set__bosize(i,mexp(lsum/denom));
387       fi;
388       i:=i-1;
389     endfor;
390   endgroup;
391 enddef;
392
393 vardef build_kanji.lstransform(expr thresh,dist,mypt) =
394   if ypart mypt>thresh:
395     mypt
396   else:
397     (xpart mypt,
398      (ypart mypt)*((thresh-dist)/thresh)
399      +(xpart mypt/1000)[(dist/thresh)*ypart mypt,dist])
400   fi
401 enddef;
402
403 vardef build_kanji.lift_skirt(expr thresh,dist)(text curves) =
404   begingroup
405     save osp;
406     numeric osp;

```

```

407   osp:=sp;
408   curves;
409   save i,boti,x,y;
410   numeric x[],y[];
411   i:=0;
412   boti:=whatever;
413   y0:=1000;
414   forever:
415     exitif find_stroke(i)<osp;
416     if (ypart llcorner get__strokep(i))=(ypart urcorner get__strokep(i)):
417       if (unknown boti) or (ypart llcorner get__strokep(i)<y0):
418         y0:=ypart llcorner get__strokep(i);
419         boti:=i;
420       fi;
421     fi;
422     replace__strokep(i)(
423       for j=0 upto length oldp-1:
424         build_kanji.lstransform(thresh,dist)(point j of oldp)
425         ..controls build_kanji.lstransform(thresh,dist)
426             (postcontrol j of oldp)
427         and build_kanji.lstransform(thresh,dist)(precontrol j+1 of oldp)..
428       endfor
429       if cycle oldp:
430         cycle
431       else:
432         build_kanji.lstransform(thresh,dist)(point infinity of oldp)
433       fi);
434     i:=i-1;
435   endfor;
436   if (known boti) and (y0<=thresh):
437     replace__strokep(boti)
438       (((0,7)+point 0 of oldp)..tension 1.2..((0,-5)+point 0.5 of oldp)..
439       ((0,10)+point 1 of oldp));
440     replace__strokeq(boti)((1,7,1,7)-(1,5,1,5)-(1,1));
441     set__boserif(boti,1,whatever);
442   fi;
443 endgroup;
444 enddef;
445
446 vardef build_kanji.lr(expr splitpoint,overlap)
447   (text leftstuff)(text rightstuff) =
448   perl__structure:=perl__structure
449   &"['build_kanji.lr",eids.u2ff0._1,1,"';
450 begingroup
451   save t;
452   transform t[];
453   yypart t1=yypart t2=1;
454   ypart t1=yxpart t1=yxpart t1=yxpart t2=yxpart t2=0;

```

BUIL

```

455 (50,0) transformed t1=(50,0);
456 (950,0) transformed t1=(splitpoint+overlap/2,0);
457 (50,0) transformed t2=(splitpoint-overlap/2,0);
458 (950,0) transformed t2=(950,0);
459 tsu_xform(t1)(leftstuff);
460 perl_structure:=perl_structure&"",[";
461 tsu_xform(t2)(rightstuff);
462 endgroup;
463 perl_structure:=perl_structure&""]";
464 enddef;
465
466 vardef build_kanji.sscale(text tran)(text curves) =
467 tsu_xform(identity shifted (-centre_pt) tran shifted centre_pt)(curves);
468 enddef;
469
470 vardef build_kanji.spread_legs(expr dist)(text curves) =
471 begingroup
472 save osp;
473 numeric osp;
474 osp:=sp;
475 curves;
476 save mytr;
477 transform mytr[];
478 (50,50) transformed mytr1=(50,50);
479 (500,50) transformed mytr1=(500-dist/2,50);
480 (500,850) transformed mytr1=(500-dist/2,850);
481 (950,50) transformed mytr2=(950,50);
482 (500,50) transformed mytr2=(500+dist/2,50);
483 (500,850) transformed mytr2=(500+dist/2,850);
484 save i;
485 i:=0;
486 forever:
487 exitif find_stroke(i)<osp;
488 if xpart 0.75[llcorner get_stroke(i),urcorner get_stroke(i)]<475:
489 replace_stroke(i,oldp transformed mytr1);
490 elseif xpart 0.25[llcorner get_stroke(i),urcorner get_stroke(i)]>525:
491 replace_stroke(i,oldp transformed mytr2);
492 fi;
493 i:=i+1;
494 endfor;
495 endgroup;
496 enddef;
497
498 vardef build_kanji.tb(expr splitpoint,overlap)
499 (text topstuff)(text bottomstuff) =
500 perl_structure:=perl_structure
501 &"['build_kanji.tb',eids.u2ffl._1l_']";
502 begingroup

```

```

503     save t;
504     transform t[];
505     xpart t1=xxpart t2=1;
506     xpart t1=ypart t1=yxpart t1=xpart t2=ypart t2=yxpart t2=0;
507     (0,850) transformed t1=(0,850);
508     (0,50) transformed t1=(0,splitpoint-overlap/2);
509     (0,850) transformed t2=(0,splitpoint+overlap/2);
510     (0,50) transformed t2=(0,50);
511     tsu_xform(t1)(topstuff);
512     perl_structure:=perl_structure&"],[";
513     tsu_xform(t2)(bottomstuff);
514 endgroup;
515 perl_structure:=perl_structure&""]];";
516 endif;
517
518 vardef build_kanji.tcb(expr splitpointa,overlapa)(expr splitpointb,overlapb)
519 (text topstuff)(text centrestuff)(text bottomstuff) =
520 perl_structure:=perl_structure
521 &"['build_kanji.tcb',eids.u2ff3._2.1_1.2_'];[";
522 begingroup
523     save t;
524     transform t[];
525     xpart t1=xxpart t2=xxpart t3=1;
526     xpart t1=ypart t1=yxpart t1=0;
527     xpart t2=ypart t2=yxpart t2=0;
528     xpart t3=ypart t3=yxpart t3=0;
529     (0,850) transformed t1=(0,850);
530     (0,50) transformed t1=(0,splitpointa-overlapa/2);
531     (0,850) transformed t2=(0,splitpointa+overlapa/2);
532     (0,50) transformed t2=(0,splitpointb-overlapb/2);
533     (0,850) transformed t3=(0,splitpointb+overlapb/2);
534     (0,50) transformed t3=(0,50);
535     tsu_xform(t1)(topstuff);
536     perl_structure:=perl_structure&"],[";
537     tsu_xform(t2)(centrestuff);
538     perl_structure:=perl_structure&"],[";
539     tsu_xform(t3)(bottomstuff);
540 endgroup;
541 perl_structure:=perl_structure&""]];";
542 endif;
543
544 vardef build_kanji.thickness(expr amount)(text curves) =
545 begingroup
546     save xsp;
547     xsp:=sp;
548     curves;
549     i:=0;
550     forever:

```

```

551     exitif find_stroke(i)<xsp;
552     if unknown get_bosize(i):
553         set_bosize(i,100);
554     fi;
555     if get_bosize(i)>0:
556         set_bosize(i,get_bosize(i)*amount);
557     fi;
558     i:=i-1;
559     endfor;
560 endgroup;
561 enddef;
562
563 vardef build_kanji.triclusterc(expr topxscale)
564 (text topstuff)(text leftstuff)(text rightstuff) =
565     build_kanji.tb(500,0)
566     (build_kanji.sscale(xscaled topxscale)(topstuff))
567     (build_kanji.lr(480,0)
568     (leftstuff)
569     (rightstuff));
570 enddef;
571
572 vardef build_kanji.wptransform(expr ca,cb,cc,cd)(expr inpt) =
573     begingroup
574         save myxf,x,y;
575         transform myxf;
576         numeric x[],y[];
577         (50,-50) transformed myxf=(0,0);
578         (50,850) transformed myxf=(0,1);
579         (950,50) transformed myxf=(1,0);
580         z1=inpt transformed myxf;
581         z2=y1[x1[cc,cd],x1[cb,ca]];
582         z2 % no semicolon
583     endgroup % no semicolon
584 enddef;
585
586 vardef build_kanji.warp(expr ca,cb,cc,cd)(text curves) =
587     begingroup
588         save osp;
589         numeric osp;
590         osp:=sp;
591         curves;
592         save i;
593         i:=0;
594         forever:
595             exitif find_stroke(i)<osp;
596             replace_stroke(i)(
597                 for j=0 upto length oldp-1:
598                     build_kanji.wptransform(ca,cb,cc,cd)(point j of oldp)

```



```

599         ..controls build_kanji.wptransform(ca,cb,cc,cd)
600                                     (postcontrol j of oldp)
601         and build_kanji.wptransform(ca,cb,cc,cd)(precontrol j+1 of oldp)..
602     endfor
603     if cycle oldp:
604         cycle
605     else:
606         build_kanji.wptransform(ca,cb,cc,cd)(point infinity of oldp)
607     fi);
608     i:=i-1;
609 endfor;
610 endgroup;
611 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,-80,10)..(-35,35)..(10,90)) transformed xf,
40     (1,1)..(14,14)..(18,18));
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)..(14,14)..(18,18));
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;
```

enclosed.mp

```
1 %
2 % Enclosed characters for Tsukurimashou
3 % Copyright (C) 2011, 2012, 2013 Matthew Skala
4 %
5-29 [Standard copyright notice]
30
31 inclusion_lock(enclosed);
32
33 

---


34
35 vardef circle.single =
36   push_stroke(fullcircle scaled 810 shifted centre_pt,
37     (2,2)-(2,2)-(2,2)-(2,2)-cycle);
38   set_bobrush(0,brpunct);
39   set_bosize(0,53);
40   tsu_render;
41   init_obstack;
42 enddef;
43
44 vardef circle.double =
45   push_stroke(fullcircle scaled 900 shifted centre_pt,
46     (2,2)-(2,2)-(2,2)-(2,2)-cycle);
47   set_bobrush(0,brpunct);
48   set_bosize(0,40);
49   push_stroke(fullcircle scaled 740 shifted centre_pt,
50     (2,2)-(2,2)-(2,2)-(2,2)-cycle);
51   set_bobrush(0,brpunct);
52   set_bosize(0,40);
53   tsu_render;
54   init_obstack;
55 enddef;
56
57 vardef square.single(text curves) =
58   tsu_xform(tsu_xf.sletter shifted tsu_xf.circ_slant_shift)(curves);
59   push_stroke(
60     ((500,790)-(100,790)-(100,-10)-(900,-10)-(900,790)-cycle)
61     transformed inverse tsu_slant_xform,
62     (2,2)-(2,2)-(2,2)-(2,2)-(2,2)-cycle);
63   set_bobrush(0,brpunct);
64   set_bosize(0,80);
65   set_botip(0,1);
66   set_botip(0,2,1);
67   set_botip(0,3,1);
68   set_botip(0,4,1);
69 enddef;
70
```

ENCL

```

71
72
73 transform tsu_xf.circled;
74 xypart tsu_xf.circled=yypart tsu_xf.circled=0.68;
75 xypart tsu_xf.circled=yxpart tsu_xf.circled=0;
76 centre_pt transformed tsu_xf.circled=centre_pt;
77
78 transform tsu_xf.cletter;
79 xypart tsu_xf.cletter=yypart tsu_xf.cletter=0.56;
80 xypart tsu_xf.cletter=yxpart tsu_xf.cletter=0;
81 centre_pt transformed tsu_xf.cletter=centre_pt;
82
83 transform tsu_xf.ctwo.left;
84 xypart tsu_xf.ctwo.left=yypart tsu_xf.ctwo.left=0.48;
85 xypart tsu_xf.ctwo.left=yxpart tsu_xf.ctwo.left=0;
86 (centre_pt+290*right) transformed tsu_xf.ctwo.left=centre_pt;
87
88 transform tsu_xf.ctwo.right;
89 xypart tsu_xf.ctwo.right=yypart tsu_xf.ctwo.right=0.48;
90 xypart tsu_xf.ctwo.right=yxpart tsu_xf.ctwo.right=0;
91 (centre_pt+310*left) transformed tsu_xf.ctwo.right=centre_pt;
92
93 transform tsu_xf.sletter;
94 xypart tsu_xf.sletter=yypart tsu_xf.sletter=0.71;
95 xypart tsu_xf.sletter=yxpart tsu_xf.sletter=0;
96 centre_pt transformed tsu_xf.sletter=centre_pt+10*up;
97
98 vardef tsu_xf.circ_slant_shift =
99   (centre_pt-(centre_pt transformed tsu_slant_xform))
100 enddef;
101
102
103
104 transform tsu_xf.cbound;
105 tsu_xf.cbound=identity scaled 340 shifted centre_pt;

```

ENCL

genjimon.mp

```
1 %
2 % Genjimon glyphs
3 % Copyright (C) 2011, 2012, 2021 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-199) 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 abs(z3-z4)<0.00001:
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_rouned 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

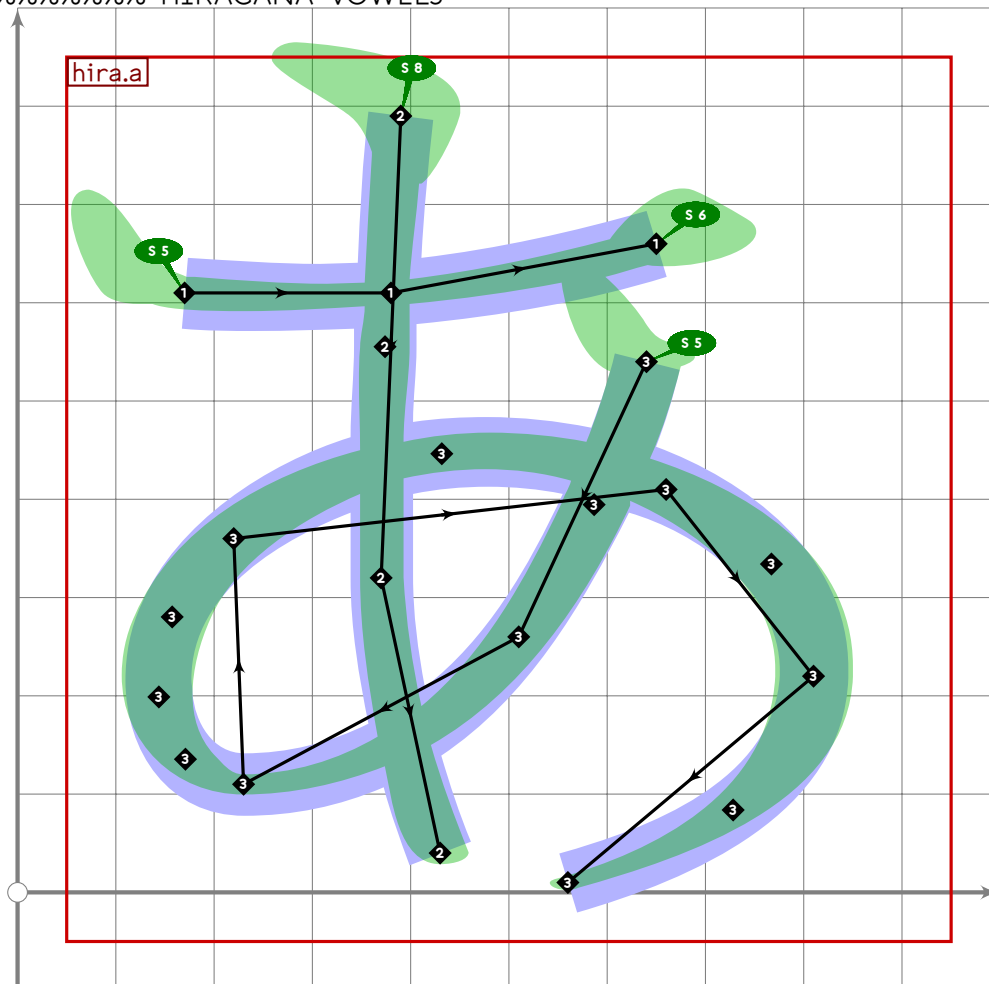
```

1 %
2 % Hiragana for Tsukurimashou
3 % Copyright (C) 2011, 2012, 2013, 2021 Matthew Skala
4 %
5-29 [Standard copyright notice]
30
31 inclusion_lock(hiragana);
32
33
34

```

Hiragana Vowels

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



HIRA

```

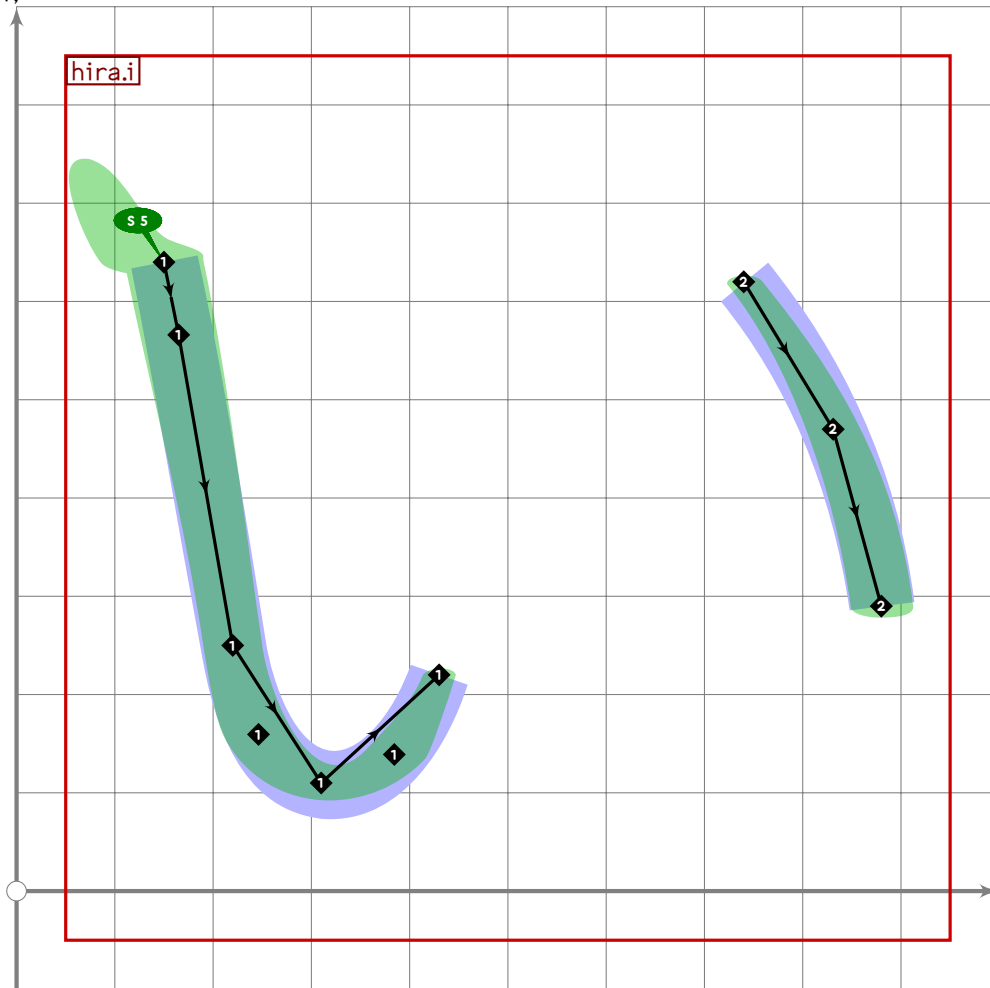
36
37 vardef hira.a =
38   push_pbox_toexpand("hira.a");
39
40   push_stroke((170,610)..(380,610)..(650,660),
41     (1,6,1,7)-(1,4,1,4)-(1,6,1,6));

```

```

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;

```



HIRA

```

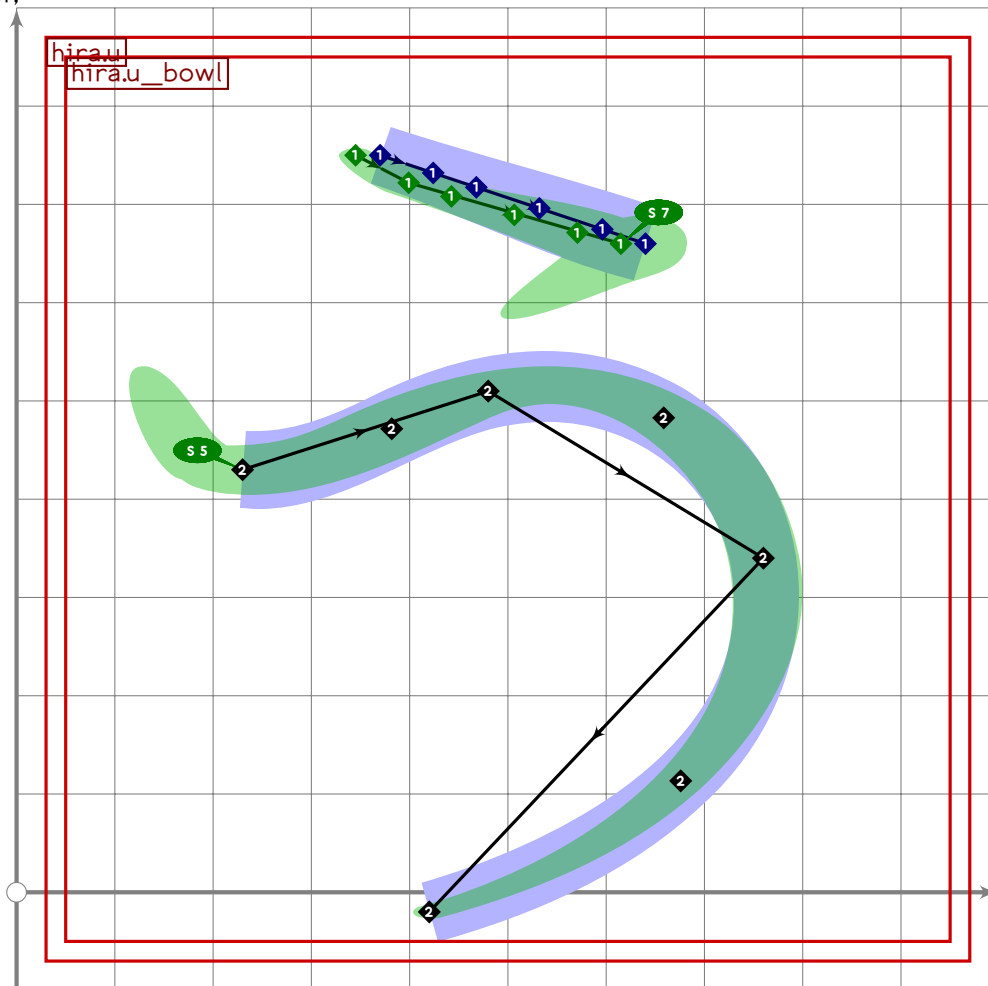
56
57 vardef hira.i =
58   push_pbox_toexpand("hira.i");
59
60   push_stroke(
61     (150,640)..(165,566)..(220,250)..(310,110)..{curl 0.1}{430,220},
62     (1.6,1.6)..(1.6,1.6)..(1.3,1.3)..(1.8,1.8)..(1,1));

```

```

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

```



```

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

```

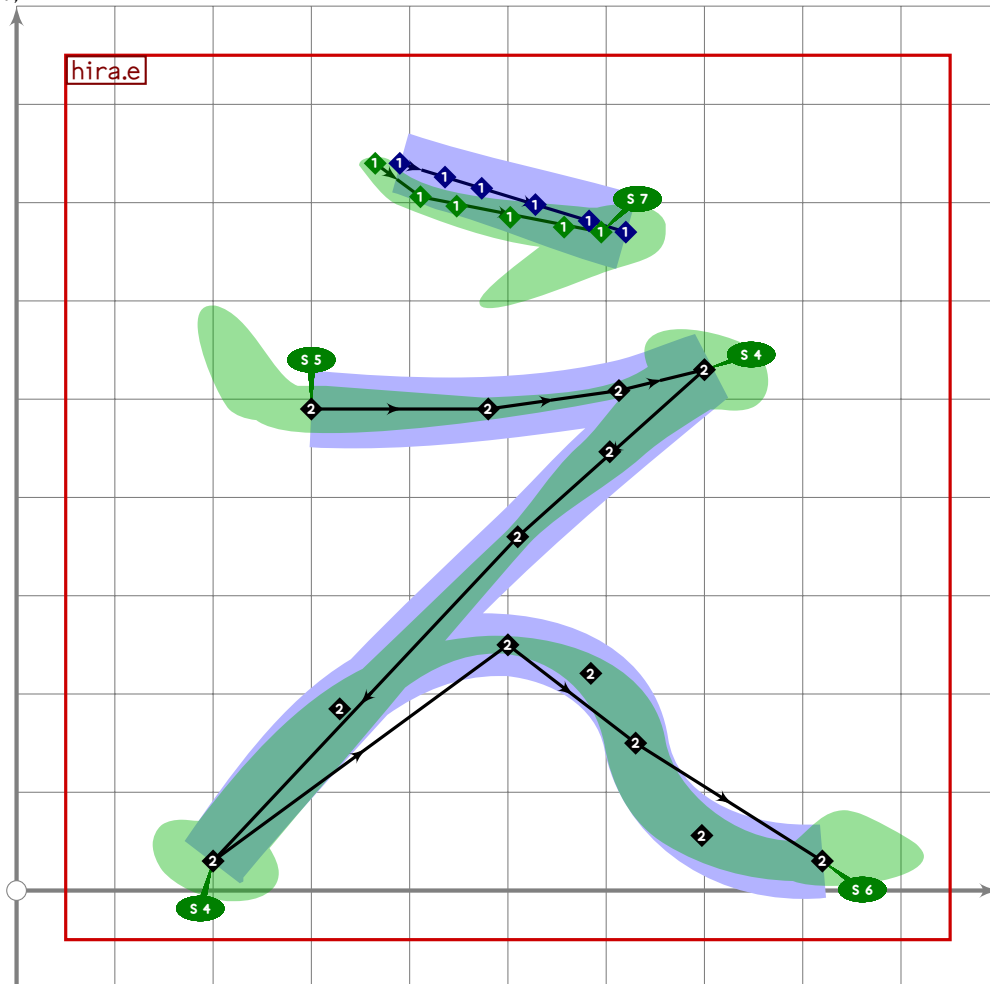
HIRA

U+3048
tsuku.uni3048

```

84 (1,1)..(14,14)..(2,3,2,3));
85 set_boserif(0,2,7);
86
87 hira.u_bowl;
88 expand_pbox;
89 enddef;

```



```

90
91 vardef hira.e =
92   push_pbox_toexpand("hira.e");
93
94   push_stroke(((390,740)..(0.2[(390,740),(620,670)]+20*down*mincho)..
95     tension 2..(620,670)) shifted (25*left*mincho,
96     (1,1)..(14,14)..(2,3,2,3));
97   set_boserif(0,2,7);
98
99   push_stroke((300,490)..(480,490)..{curl 1}{(700,530){curl 1}..
100     (510,360)..{curl 1}{(200,30){curl 0}..
101     (500,250)..(630,150){dir 280}{dir 5}{(820,30),
102     (2,2,2,2)-(1,3,1,3)-(1,1)-(2,0,1,2,0,1)
103     -(1,1)-(1,7,1,7)
104     -(1,2,1,2)-(14,14)-(2,2));

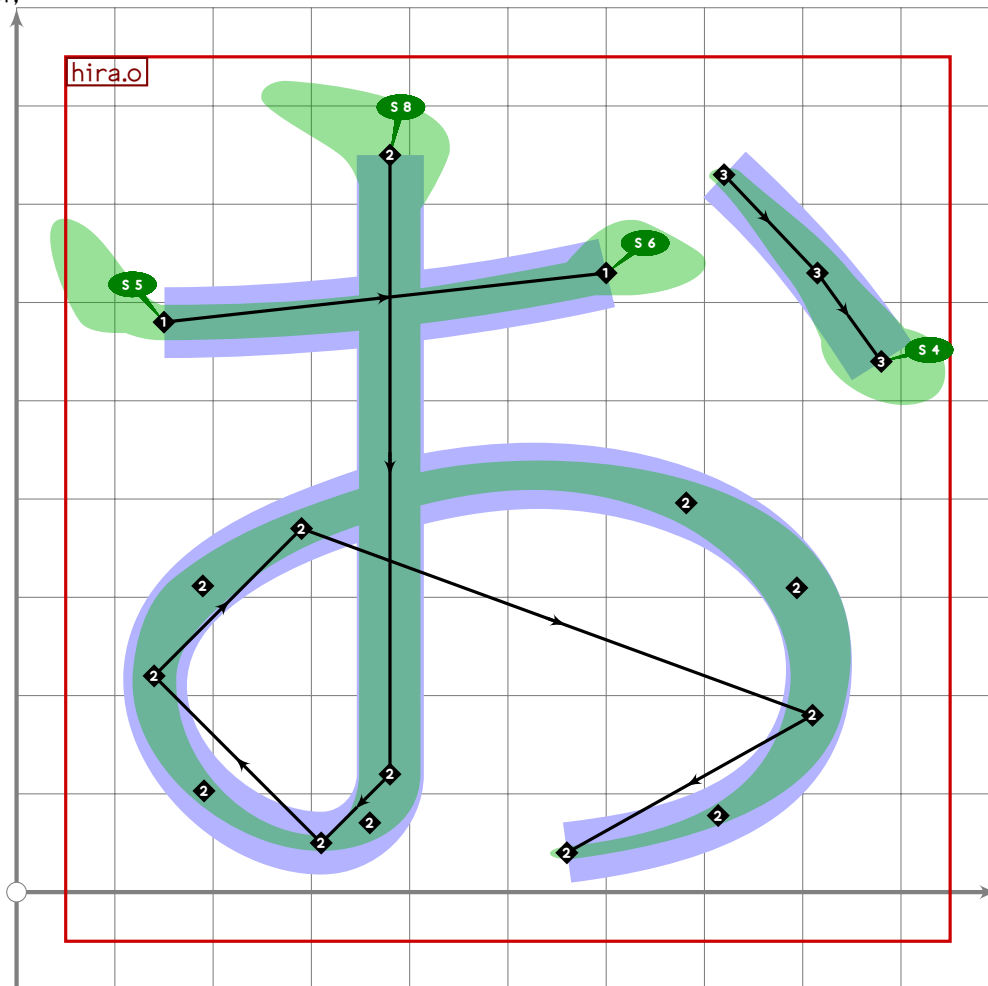
```

HIRA

```

105 replace_strokep(0)(insert_nodes(olddp)(1.6));
106 set_botip(0,3,0);
107 set_botip(0,5,0);
108 set_boserif(0,0,5);
109 set_boserif(0,3,4);
110 set_boserif(0,5,4);
111 set_boserif(0,8,6);
112 expand_pbox;
113 enddef;

```



```

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

```

HIRA

U+304B
tsuku.uni304B

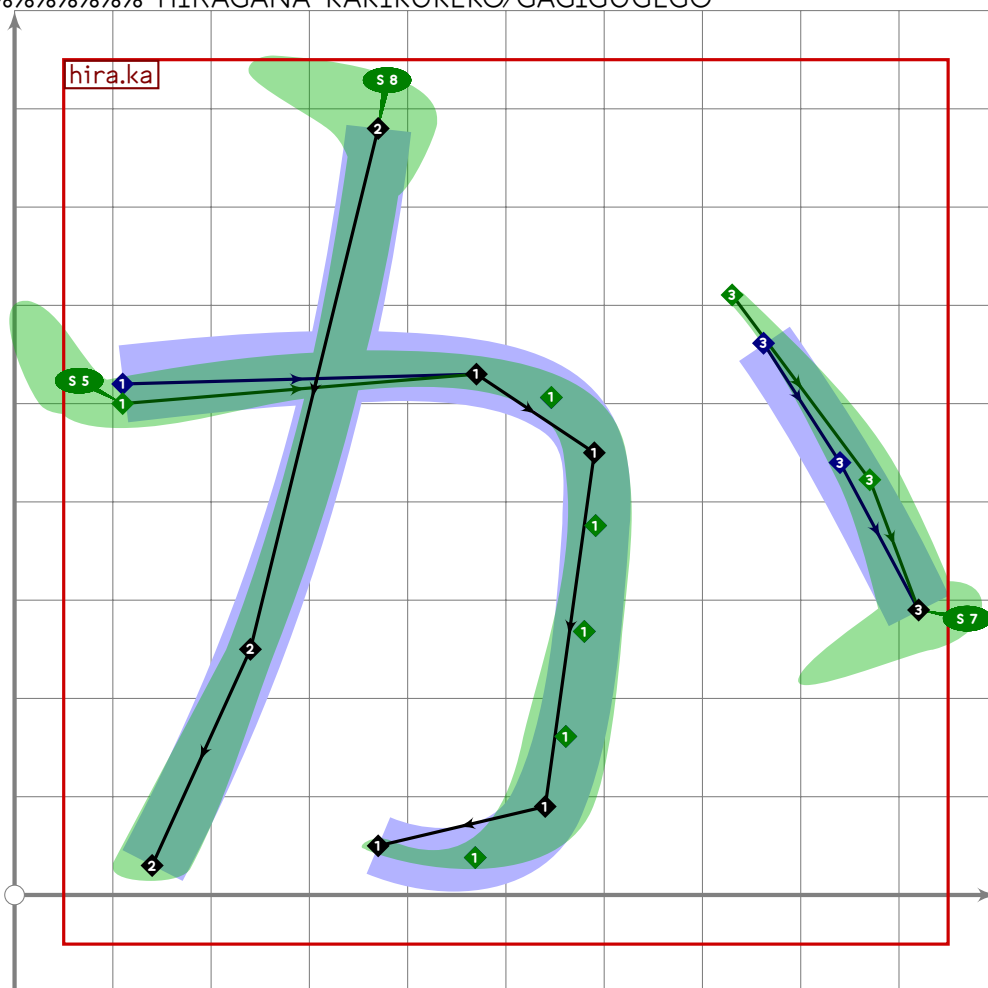
```

126      (1.6,1.6)..(1,1));
127  set__boserif(0,0,8);
128
129  push__stroke((720,730)..(815,630)..(880,540),
130      (1,1)..(14,14)..(1.8,1.8));
131  set__boserif(0,2,4);
132  expand__pbox;
133 enddef;
134

```

Hiragana Kakikukeko/Gagigugego

135 %%%%%%%%% HIRAGANA KAKIKUKEKO/GAGIGUGEGO



HIRA

```

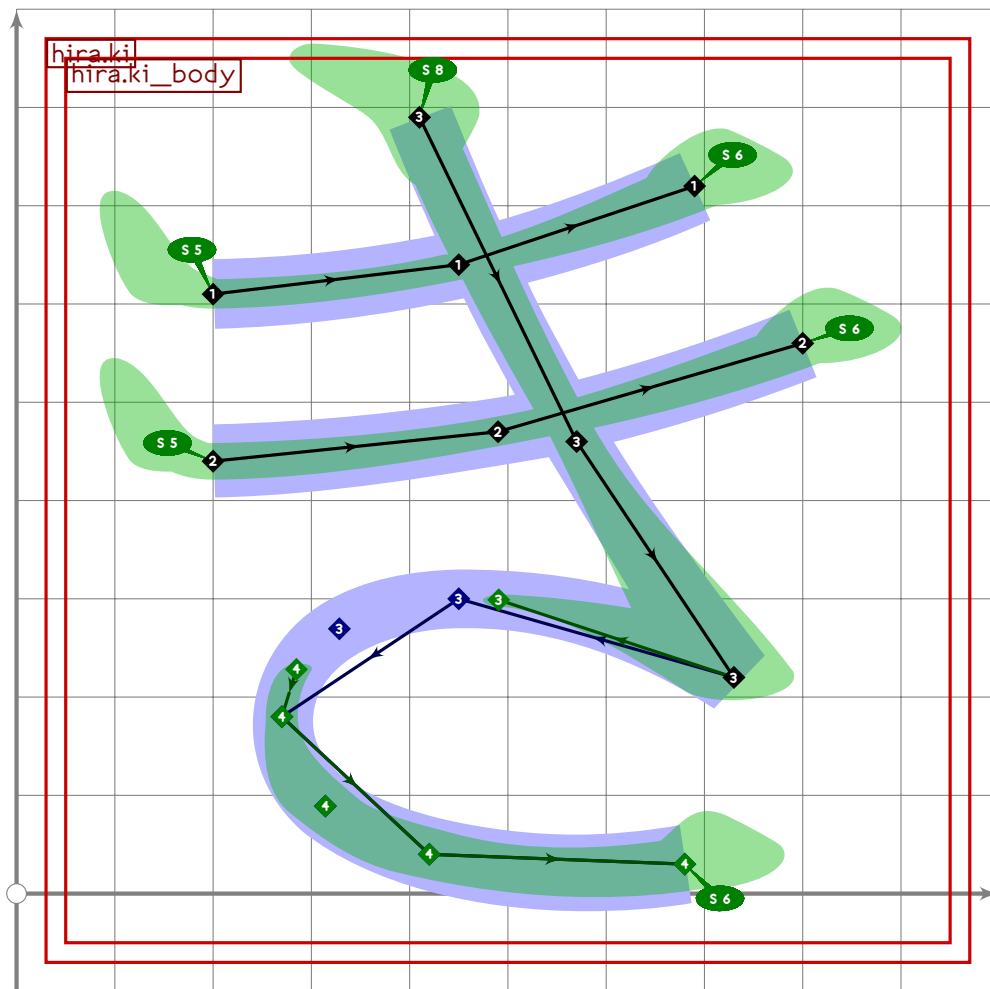
136
137 vardef hira.ka =
138   push__pbox__toexpand("hira.ka");
139
140   push__stroke(((110,520)+20*mincho*down){curl 0}..(470,530)..(590,450)..
141       tension 2..(540,90)..{curl 0.3}(370,50),
142       (2.3,2.3)..(1.7,1.7)..(14,14)..(1.8,1.8)..(1,1));
143   set__boserif(0,0,5);

```

```

144
145 push_stroke((370,780)..(240,250)..(140,30),
146   (1.3,1.3)..(1.2,1.2)..(1.6,1.6));
147 set_boserif(0,0,8);
148
149 push_stroke((720,620)..((840,440)+35*mincho*(dir -30))..(920,290),
150   (0.8,0.8)..(1.4,1.4)..(1.6,1.6));
151 set_boserif(0,2,7);
152 expand_pbox;
153 endif;
154
155 vardef hira.ki_body =
156   push_pbox__toexpand("hira.ki_body");
157
158   push_stroke((410,790)..(570,460)..{curl 1}(730,220){curl 1}..
159     (450,300)..(270,180)..(420,40)..(680,30),
160     (1.4,1.4)-(1.2,1.2)-(2.3,2)-(0.60,1)..(0.9,1.1)..
161     (2.1,2.1)..(2.4,2.4));
162   set_botip(0,2,0);
163   set_boserif(0,0,8);
164   set_boserif(0,6,6);
165   expand_pbox;
166 endif;

```

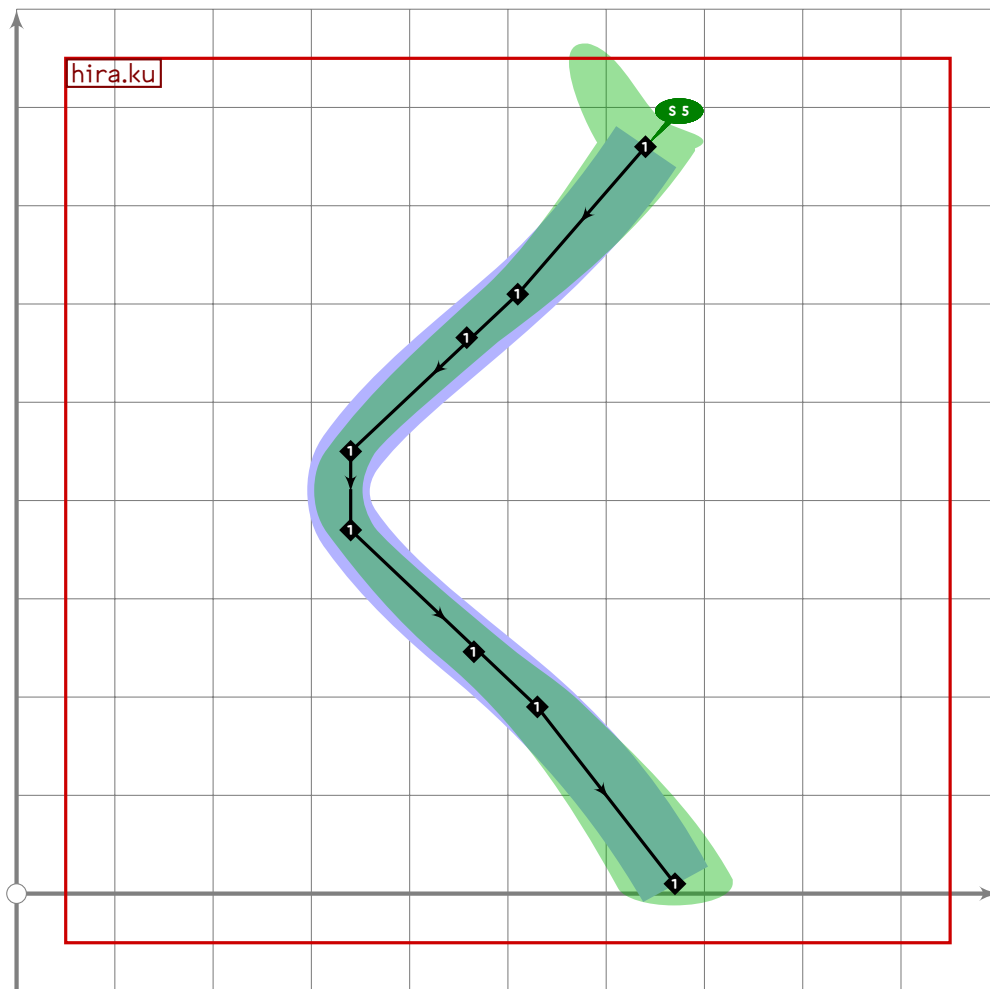


```

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

```

HIRA

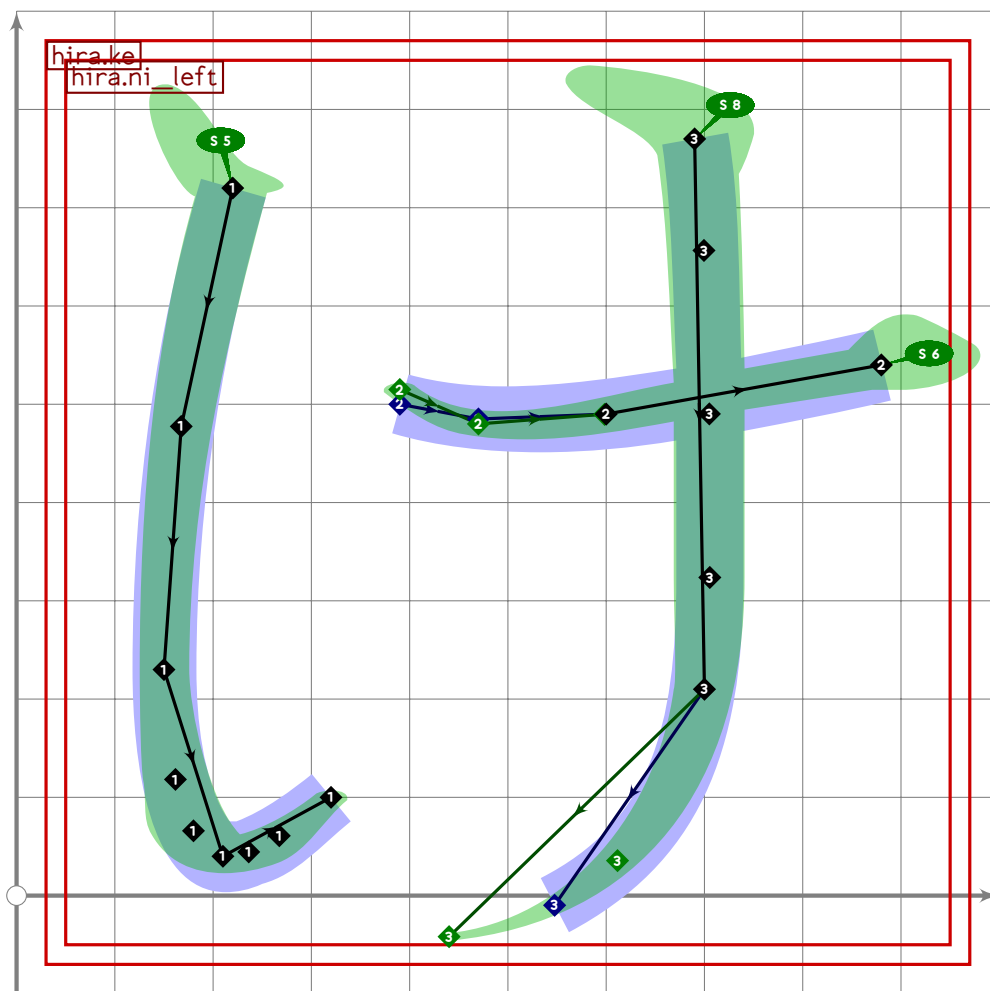


```

184
185 vardef hira.ku =
186   push_pbox__toexpand("hira.ku");
187
188   push_stroke((640,760)..(510,610)..(340,450)..
189     tension 0.75..(340,370)..(530,190)..(670,10),
190     (19,19)..(1,6,1,6)..(1,2,1,2)..(1,2,1,2)..(1,7,1,7)..(2,1,2,1));
191   set_boserif(0,0,5);
192   expand_pbox;
193 enddef;

```

HIRA

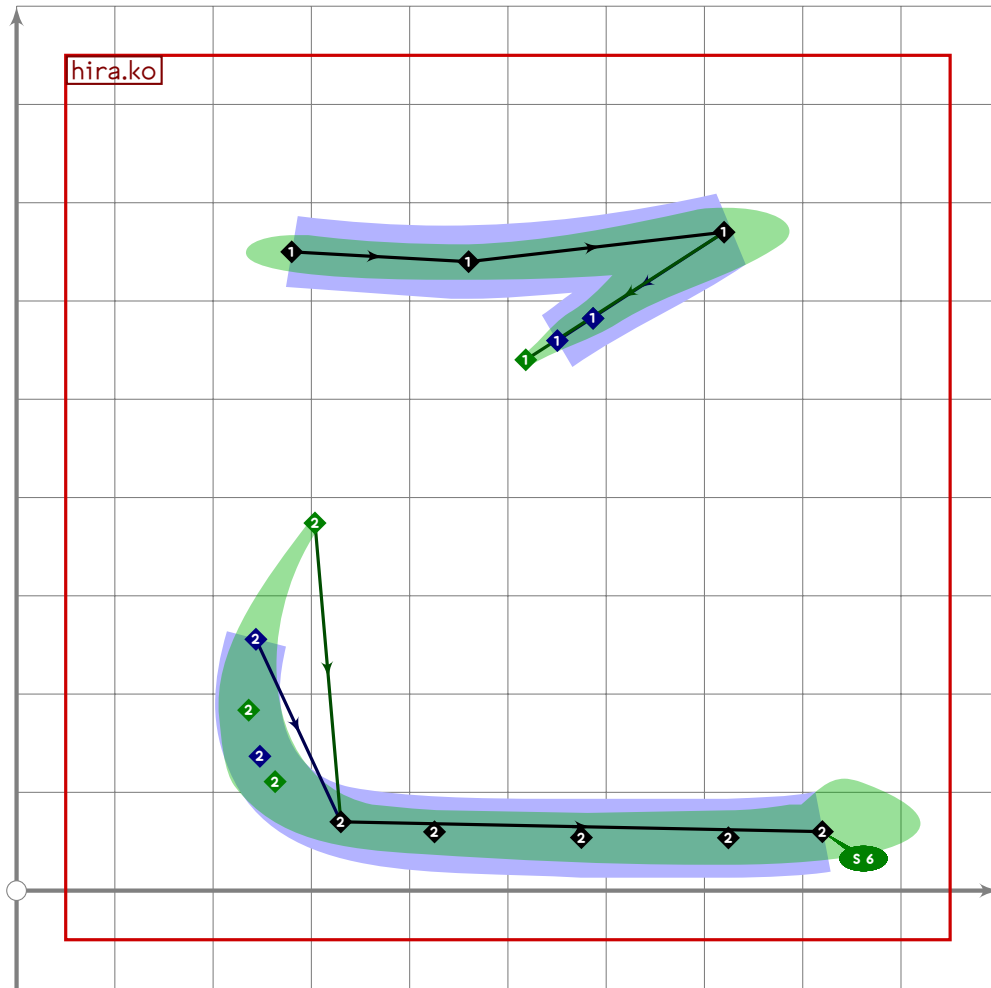


```

194
195 vardef hira.ke =
196   push_pbox_toexpand("hira.ke");
197
198   hira.ni_left;
199
200   push_stroke((390,500+15*mincho)..(470,485-5*mincho)..(600,490)..(880,540),
201     (1,1)..(14,14)..(18,18)..(2,2));
202   set_boserif(0,3,6);
203
204   push_stroke((690,770)..tension 2..(700,210)..(280,-10),
205     (1.6,1.6)-(14,14)-(0.6,0.6));
206   set_boserif(0,0,8);
207   expand_pbox;
208 enddef;

```

HIRA



```

209
210 vardef hira.ko =
211   push_pbox__toexpand("hira.ko");
212
213   push_stroke((280,650)..(460,640)..{curl 1}{720,670}{curl 1}..
214     (450,500)..(330,70)..tension 24..(820,60),
215     (1.8,1.8)-(1.9,1.9)-(2.3,2.3)-
216     (0.35,0.25)-(2.2,1.8)..(2.8,2.4));
217   set_botip(0,2,0);
218   set_boserif(0,5,6);
219   expand_pbox;
220 enddef;
221

```

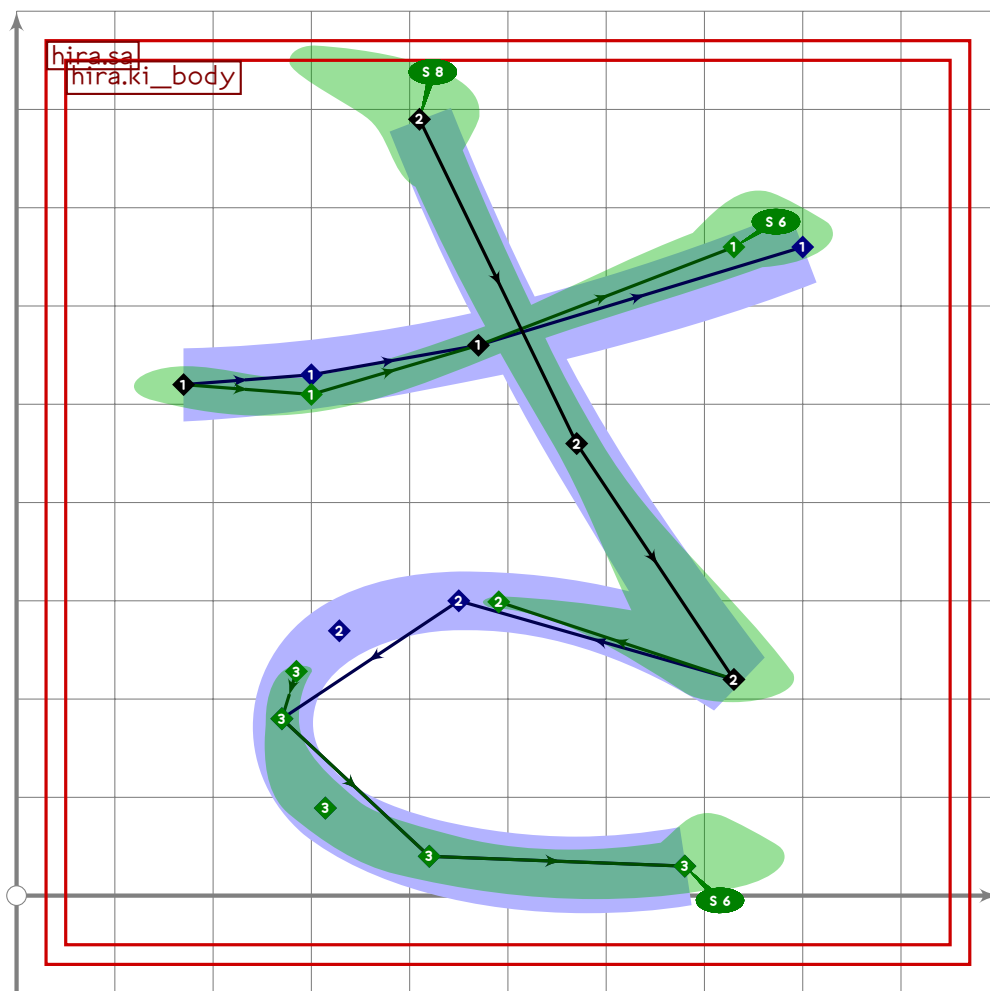
HIRA

Hiragana Sashisuseso/Zajizuzezo

```

222 %%%%%%%%% HIRAGANA SASHISUSES0/ZAJIZUZEZO

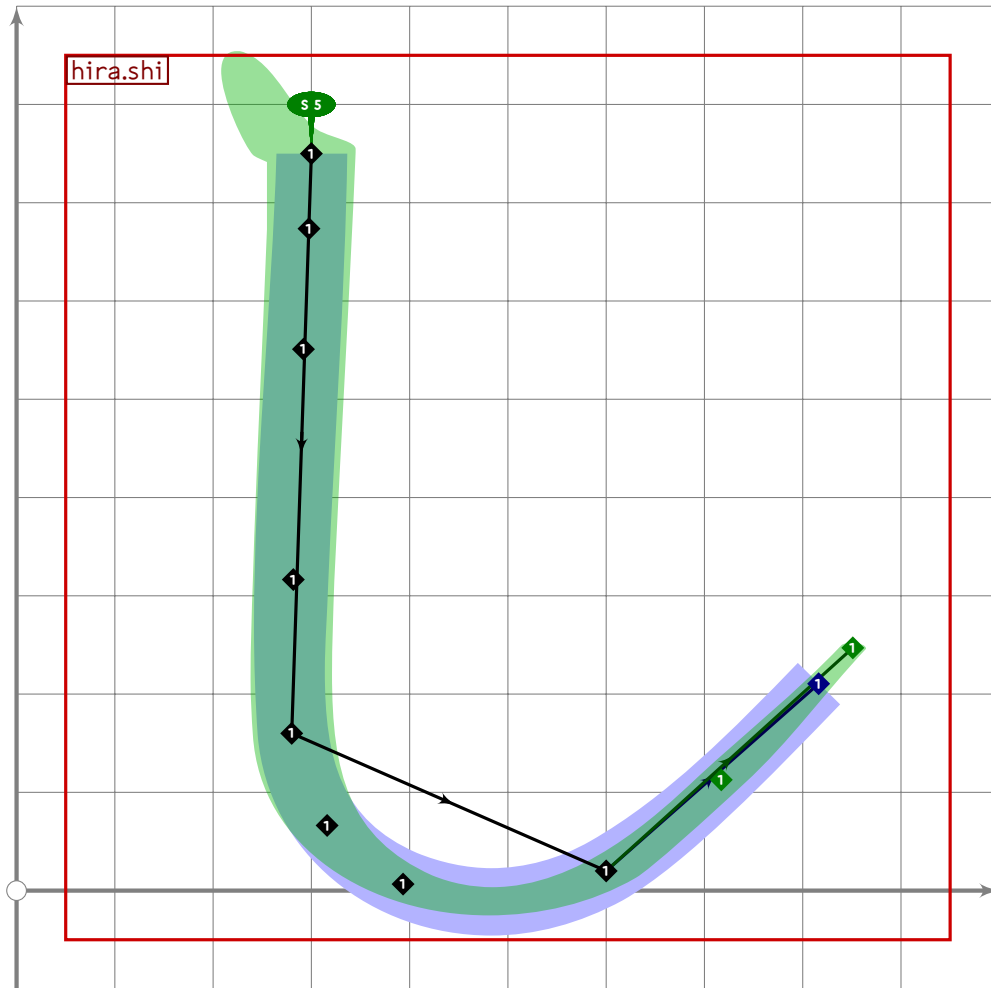
```



```

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

```

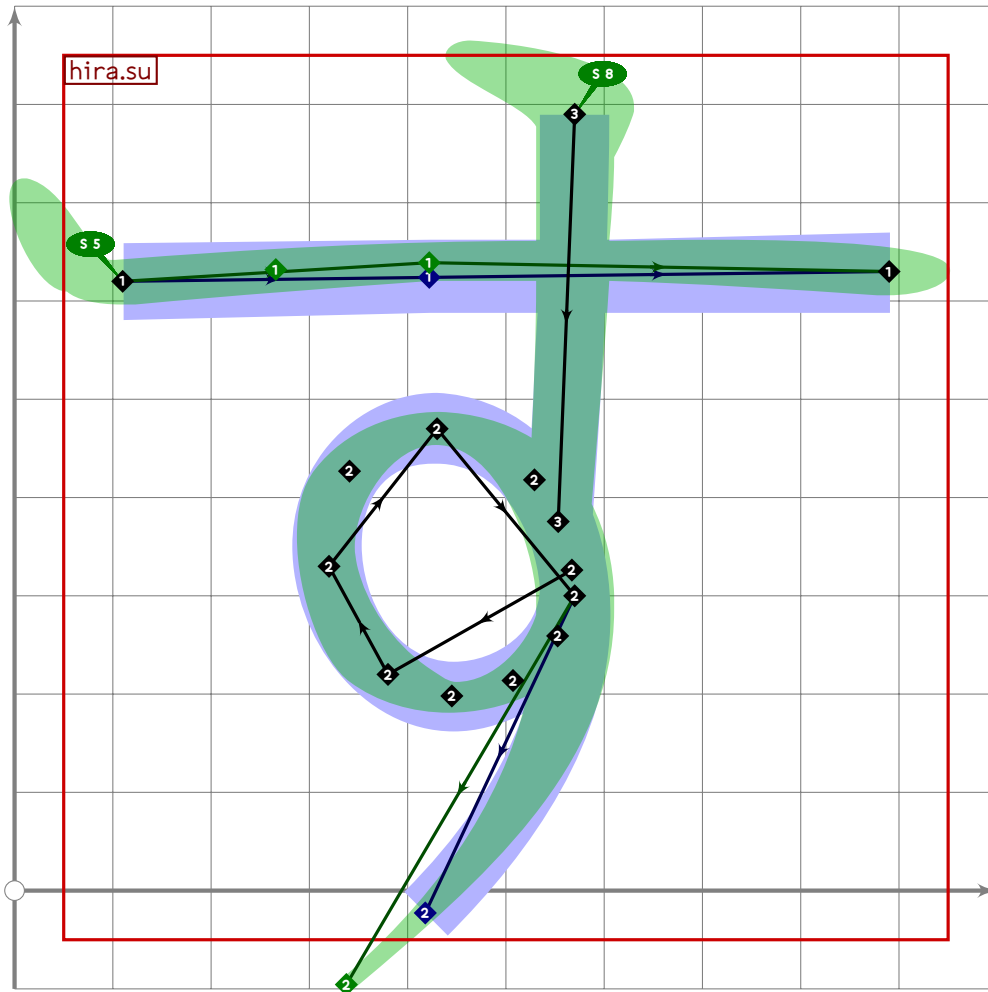


```

234
235 vardef hira.shi =
236   push_pbox__toexpand("hira.shi");
237
238   push_stroke((300,750){down}..tension 2.5..(280,160)..
239     (600,20)..tension 1.5..{curl 0}(990,400),
240     (1,7,1,7)..(1,6,1,6)-(1,5,1,5)-(0,4,0,55));
241   set_boserif(0,0,5);
242   expand_pbox;
243 enddef;

```

HIRA

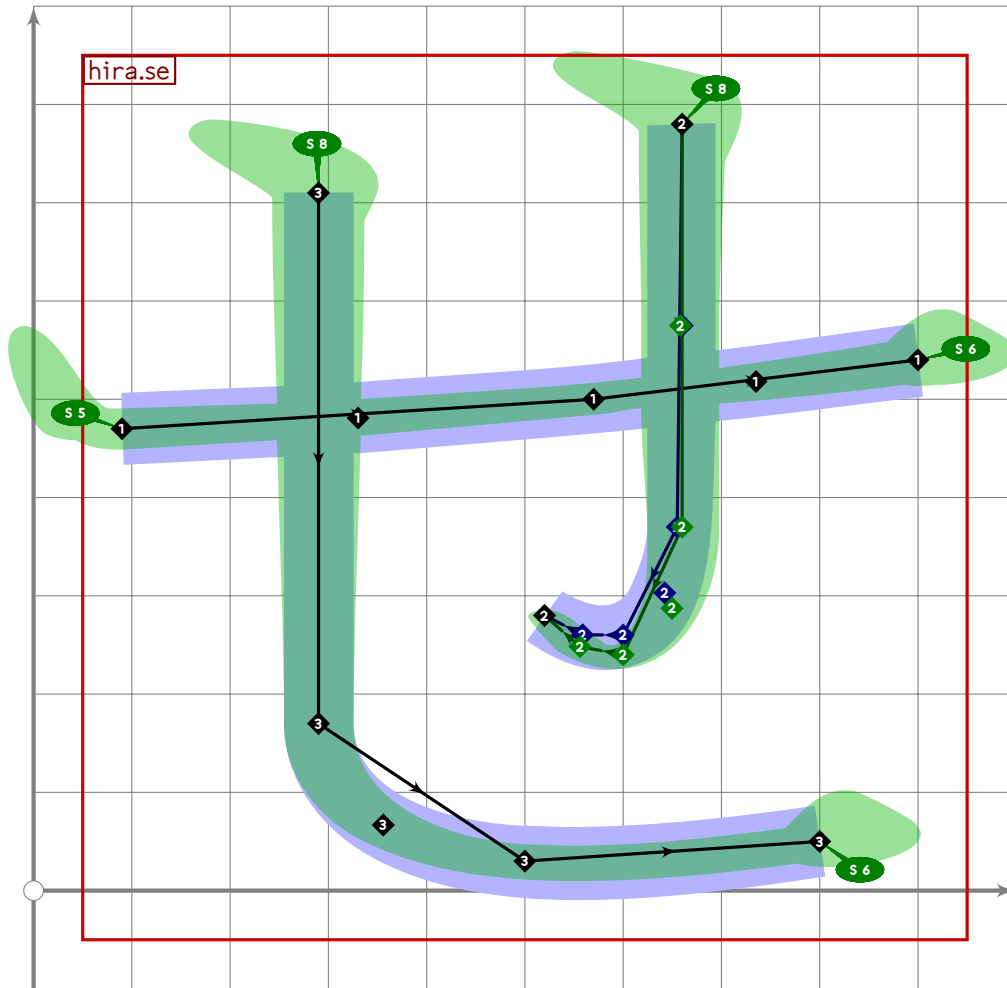


```

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

```

HIRA

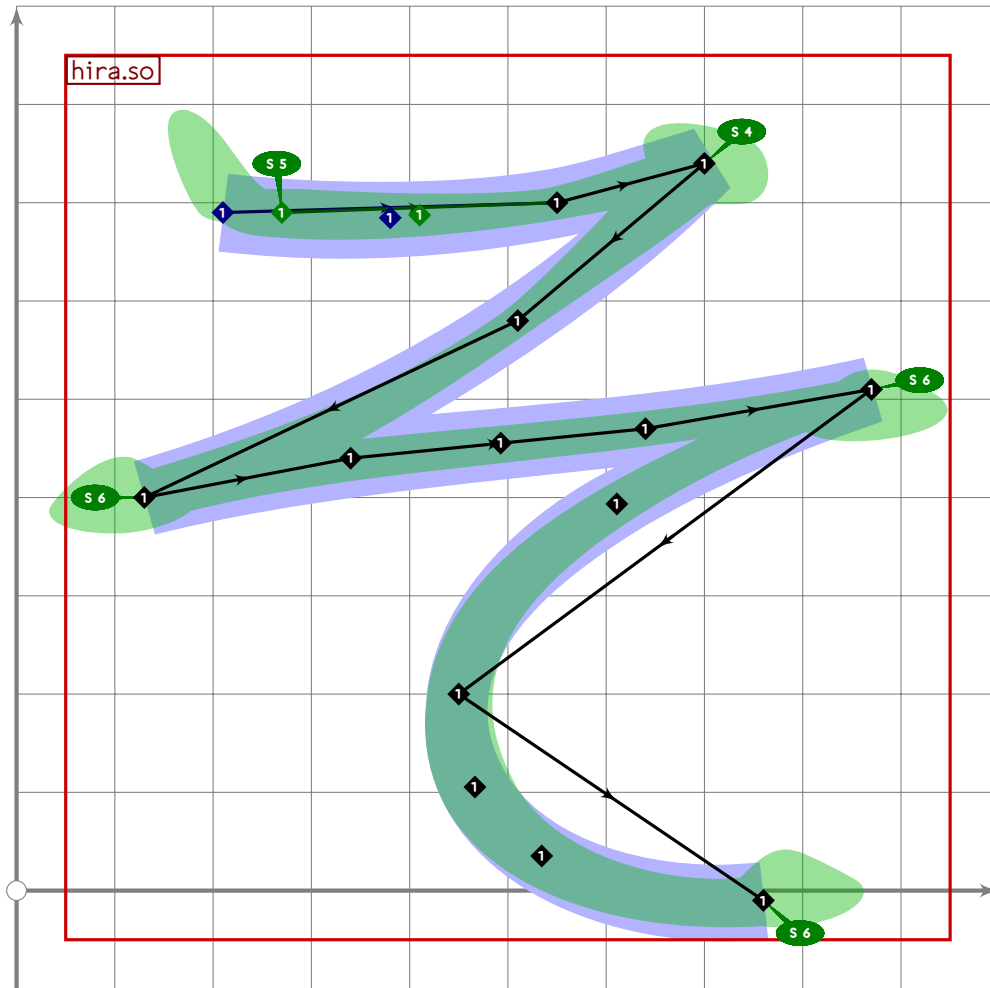


```

261
262 vardef hira.se =
263   push_pbox_toexpand("hira.se");
264
265   push_stroke((90,470)..(570,500)..(900,540),
266     (2,2)..(1.6,1.6)..(2,2));
267   set_boserif(0,0,5);
268   set_boserif(0,2,6);
269
270   push_stroke(insert_nodes((660,780)..tension 1.5..(655+5*mincho,370)..
271     (600,260-20*mincho)..{curl 0.2}(520,280))(2.5),
272     (1.7,1.7)-(1.5,1.5)-(1.3,1.3)-(1.2,1.2)-(1,1));
273   set_boserif(0,0,8);
274
275   push_stroke((290,710)..(290,170){down}..(500,30)..
276     {direction infinity of get_stroke(-1)}(800,50),
277     (1.8,1.8)-(1.5,1.5)-(1.7,1.7)-(1.8,1.8));
278   set_boserif(0,0,8);
279   set_boserif(0,3,6);
280   expand_pbox;
281 enddef;

```

HIRA



```

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

```

HIRA


```

303 set_boserif(0,6);
304 expand_pbox;
305 enddef;
306

```

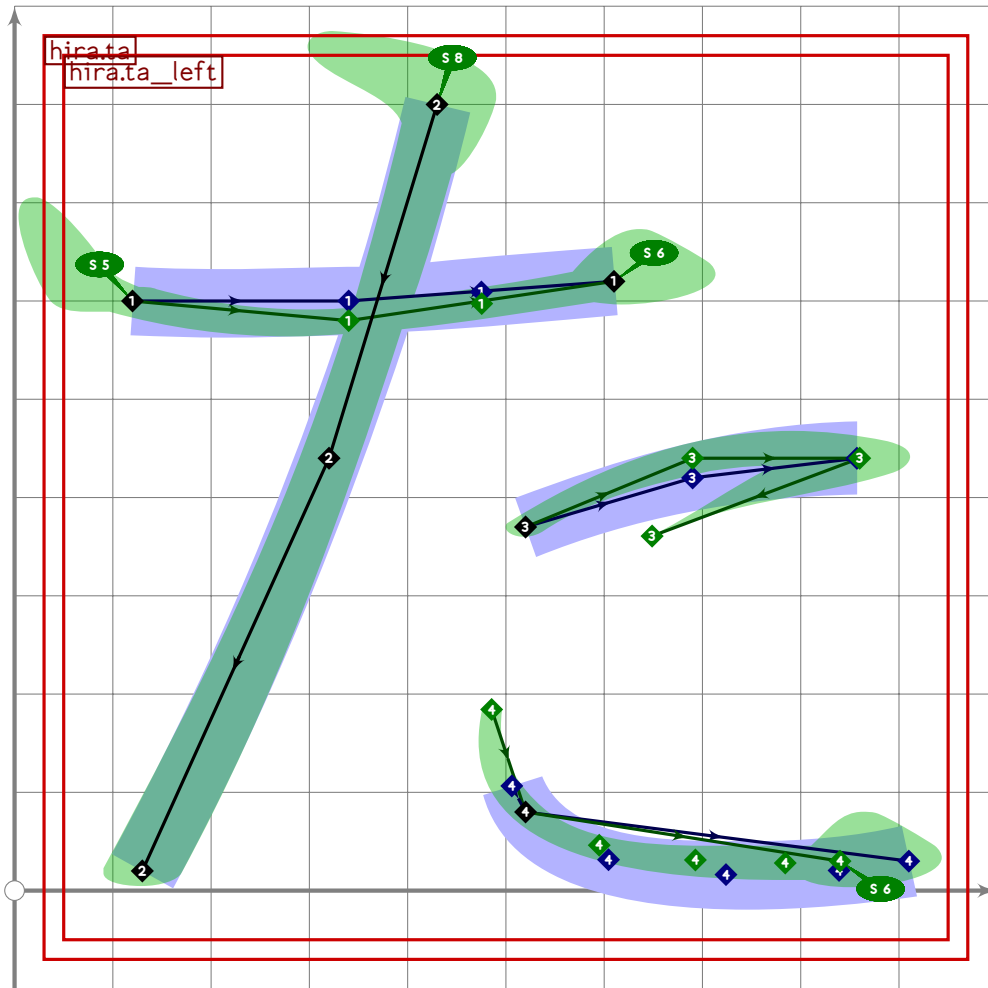
Hiragana Tachitsuteto/Dajizudedo

```

307 %%%%%%%%%% HIRAGANA TACHITSUTETO/DAJIZUDED0
308
309 vardef hira.ta_left =
310   push_pbox_toexpand("hira.ta_left");
311
312   push_stroke((120,600)..(340,600-20*mincho)..tension 1.5..(610,620),
313     (1.6,1.6)..(1.5,1.5)..(1.7,1.7));
314   set_boserif(0,0,5);
315   set_boserif(0,2,6);
316
317   push_stroke((430,800)..(320,440)..(130,20),
318     (1.4,1.4)..(1.3,1.3)..(1.6,1.6));
319   set_boserif(0,0,8);
320   expand_pbox;
321 enddef;

```

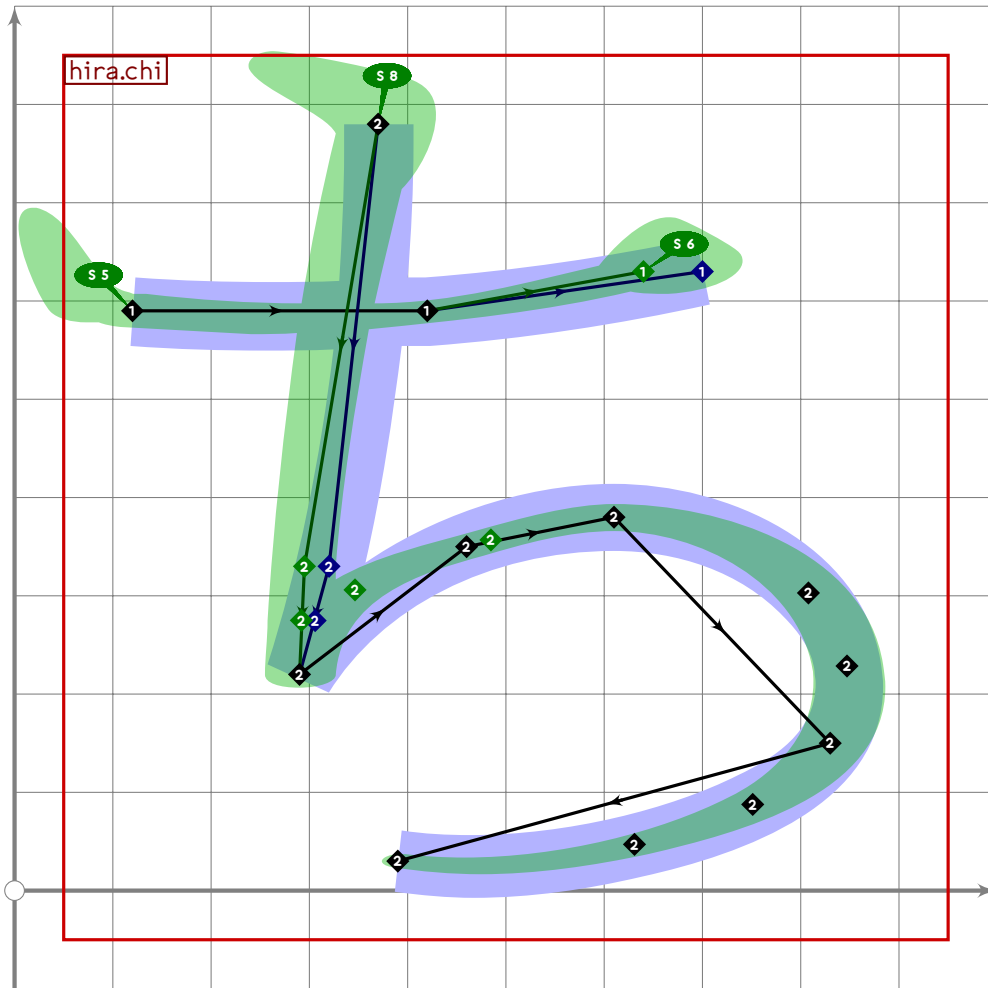
HIRA



```

322
323 vardef hira.ta =
324   push_pbox_toexpand("hira.ta");
325
326   hira.ta_left;
327
328   push_stroke((520,370)..(690,420+20*mincho)..{curl 1.5}(860,440){curl 0}..
329     (610,280+60*mincho)..(520,80)..tension 1.2 and 3..
330     {curl 0.2}(910-70*mincho,30),
331     (1,1,1)-(1.6,1.6)-(2.8,0.99)-(0.45,0.35)-(1.1,1.1)-(1.9,1.9));
332   set_botip(0,2,0);
333   set_boserif(0,5,6);
334   expand_pbox;
335 enddef;
336
337 vardef hira.chi_bottom =
338   replace_strokep(0)((oldp){-direction infinity of oldp xscaled 2}..
339     (460,350)..(610,380){right}..(830,150)..
340     tension 14..{curl 0.3}(390,30));
341   replace_strokeq(0)((oldq)..(1.3,1.3)..(1.5,1.5)..(1.5,1.5)..(1,1));
342 enddef;

```

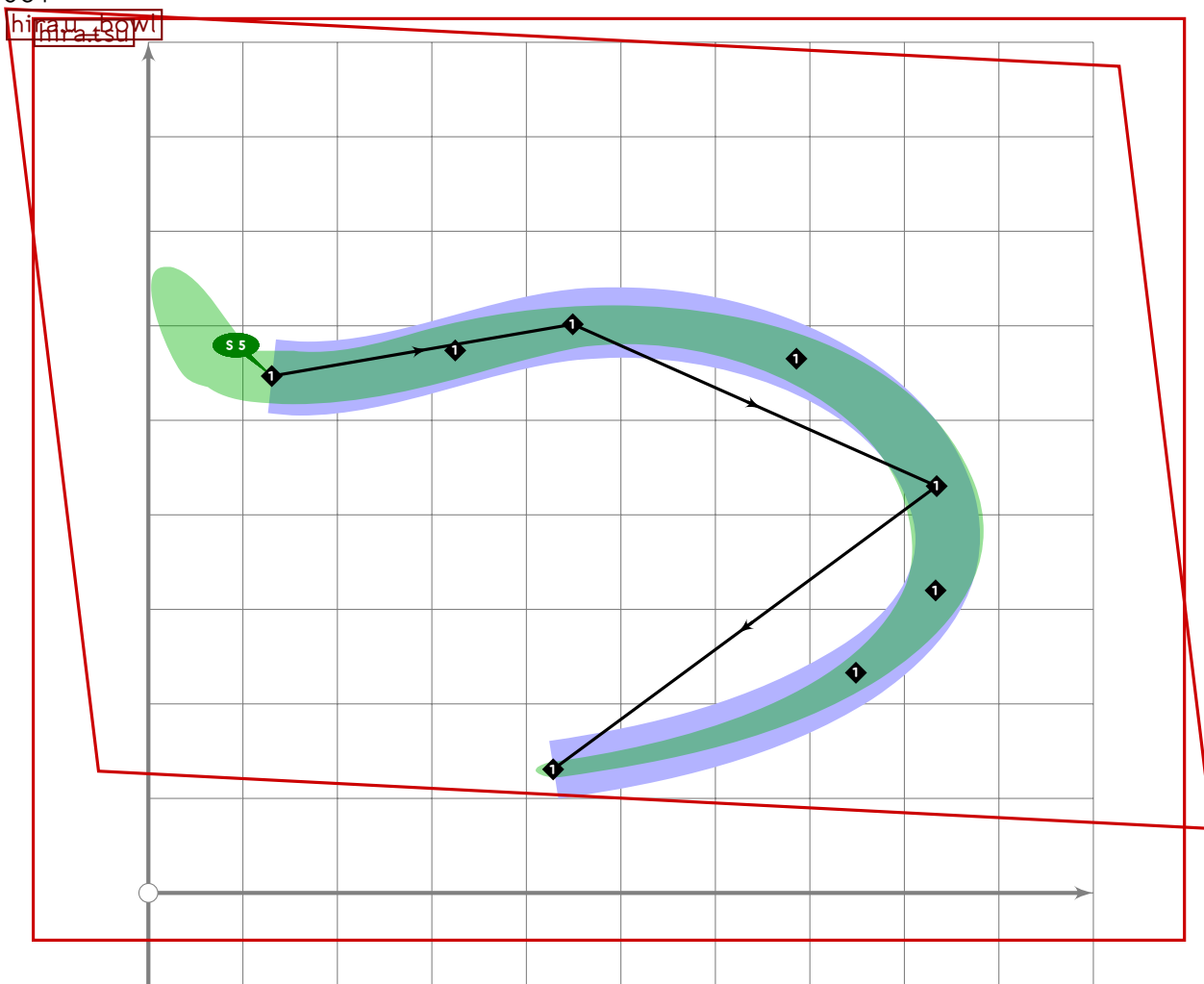


```

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

```

HIRA

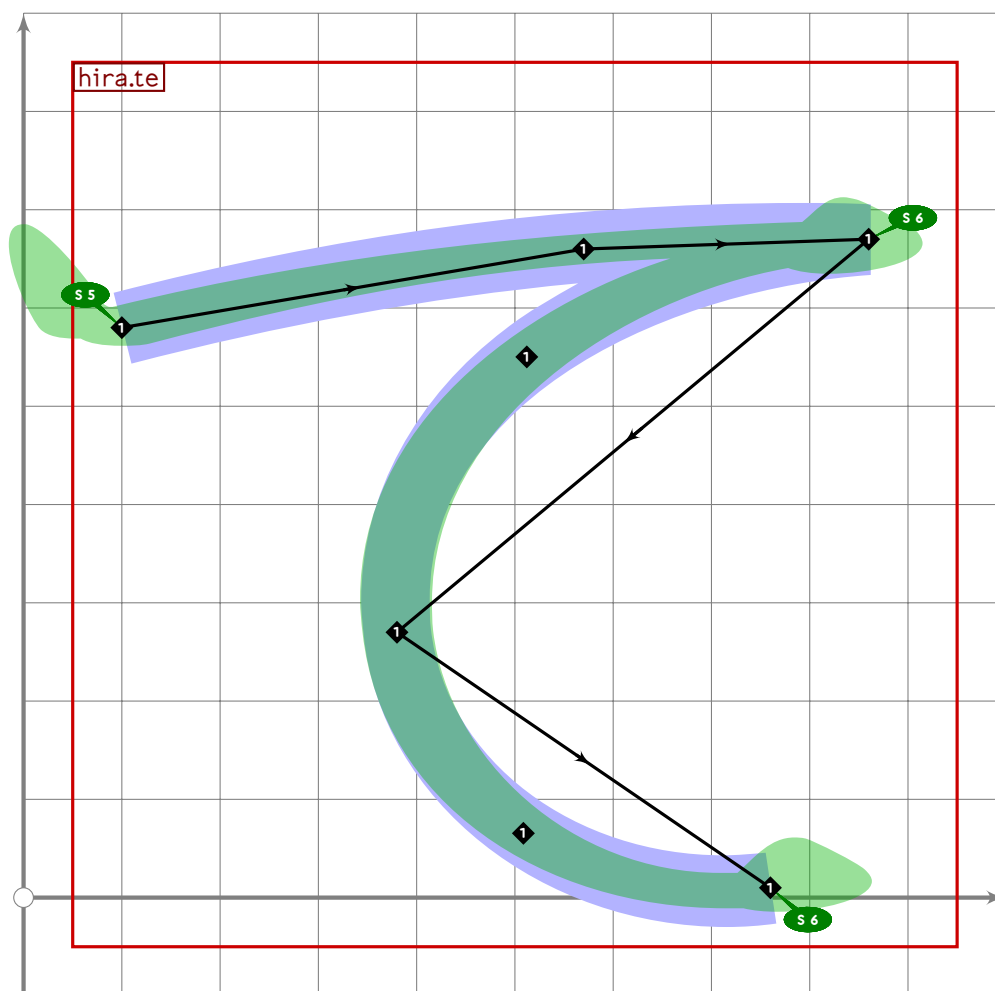


```

359
360 vardef hira.tsu =
361   push_pbox_toexpand("hira.tsu");
362
363   begingroup
364     save xf;
365     transform xf;
366     (300,450) transformed xf=(220,560);
367     (750,350) transformed xf=(820,440);
368     (400,0) transformed xf=(400,150);
369     tsu_xform(xf)(hira.u_bowl);
370     set_bosize(0)(100+10*mincho);
371   endgroup;
372   expand_pbox;
373 enddef;

```

HIRA

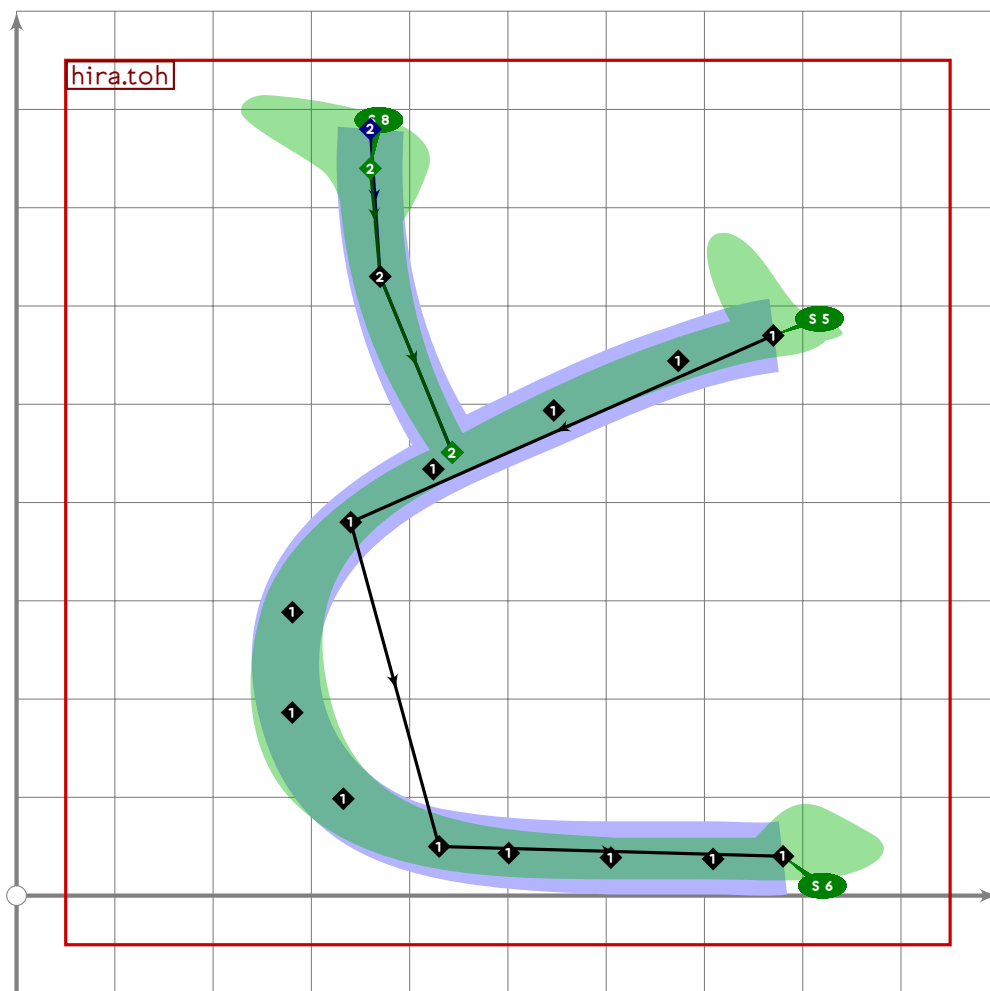


```

374
375 vardef hira.te =
376   push_pbox_toexpand("hira.te");
377
378   push_stroke((100,580)..(570,660)..{curl 1}{860,670}{curl 0.2}..
379     (380,270)..{curl 0.6}{760,10},
380     (19,19)-(1.5,1.5)-(1.8,1.8)-(1.5,1.5)-(1.8,1.8));
381   set_botip(0,2,0);
382   set_boserif(0,0,5);
383   set_boserif(0,2,6);
384   set_boserif(0,4,6);
385   expand_pbox;
386 enddef;

```

HIRA



```

387
388 vardef hira.toh =
389   push_pbox_toexpand("hira.toh");
390
391   push_stroke((770,570)..tension 1.7..(340,380)..(430,50)..
392     tension 2..(780,40),
393     (2,2)-(1.2,1.2)-(2.1,2.1)-(2,2));
394   set_boserif(0,0,5);
395   set_boserif(0,3,6);
396
397   push_stroke(subpath (0,197) of
398     ((360,780-40*mincho)..(370,630)..(point 0.7 of get_strokep(0))),
399     (1.4,1.4)-(1.3,1.3)-(1.1,1.1));
400   set_boserif(0,0,8);
401   expand_pbox;
402 enddef;
403

```

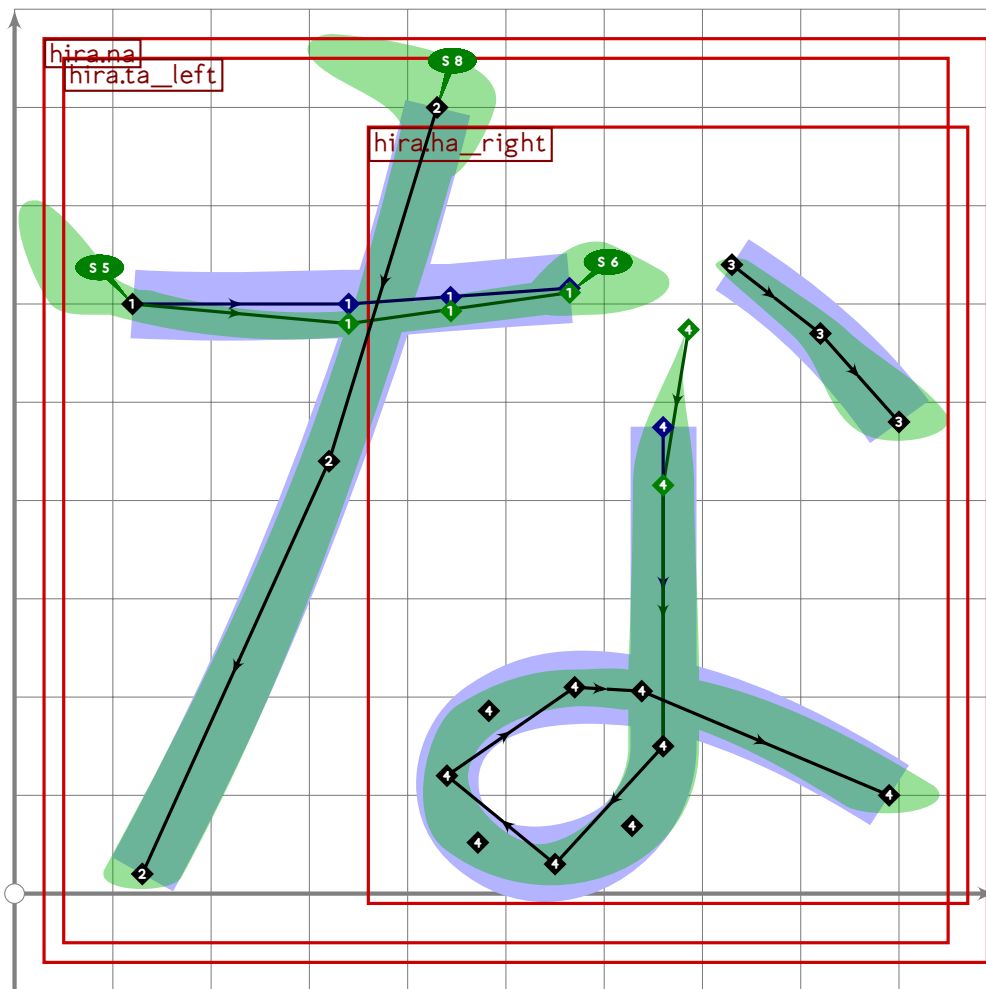
HIRA

Hiragana Naninuneno

```

404 %%%%%%%%%% HIRAGANA NANINUNENO

```



```

405
406 vardef hira.na =
407   push_pbox_toexpand("hira.na");
408
409   hira.ta_left;
410   replace_stroke(-1)(subpath (0,1.8) of oldp);
411
412   push_stroke((730,640)..(820,570)..(900,480),
413     (1,1)..(1.3,1.3)..(1.9,1.9));
414
415   hira.ha_right;
416
417   replace_stroke(0)((120,0)+point 0 of oldp)
418     ..(subpath (0.45+0.1*mincho,infinity) of oldp));
419   replace_stroke(0)((-0.2,-0.4)-(1.7,1)-(1.5,1.5)-(2,2)-
420     (1.1,1.1)-(1.9,1.9));
421   expand_pbox;
422 enddef;
423
424 vardef hira.ni_left =
425   push_pbox_toexpand("hira.ni_left");

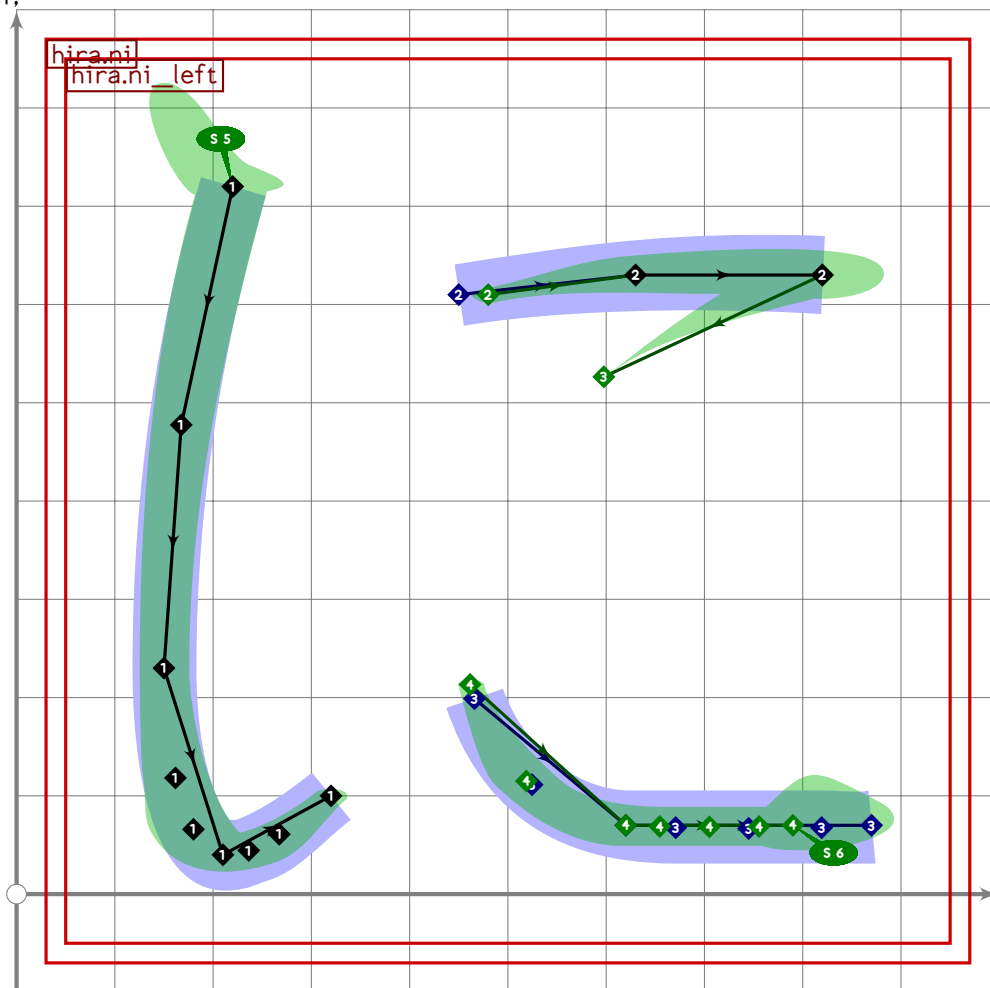
```

HIRA

```

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

```



HIRA

```

435
436 vardef hira.ni =
437   push_pbox__toexpand("hira.ni");
438
439   hira.ni_left;
440
441   push_stroke((450+30*mincho,610)..(630,630)..(820,630),
442     (1,1)..(1.9,1.9)..(2.2,2.2));
443
444   push_stroke((point infinity of get_strokep(0)){curl 0.6}..(530,460)..
445     (460,220)..(620,70)..tension 2..(870-80*mincho,70),
446     (3,0.7)-(0,0)-(0.8,0.9)-(2,2)-(1.9,1.9));

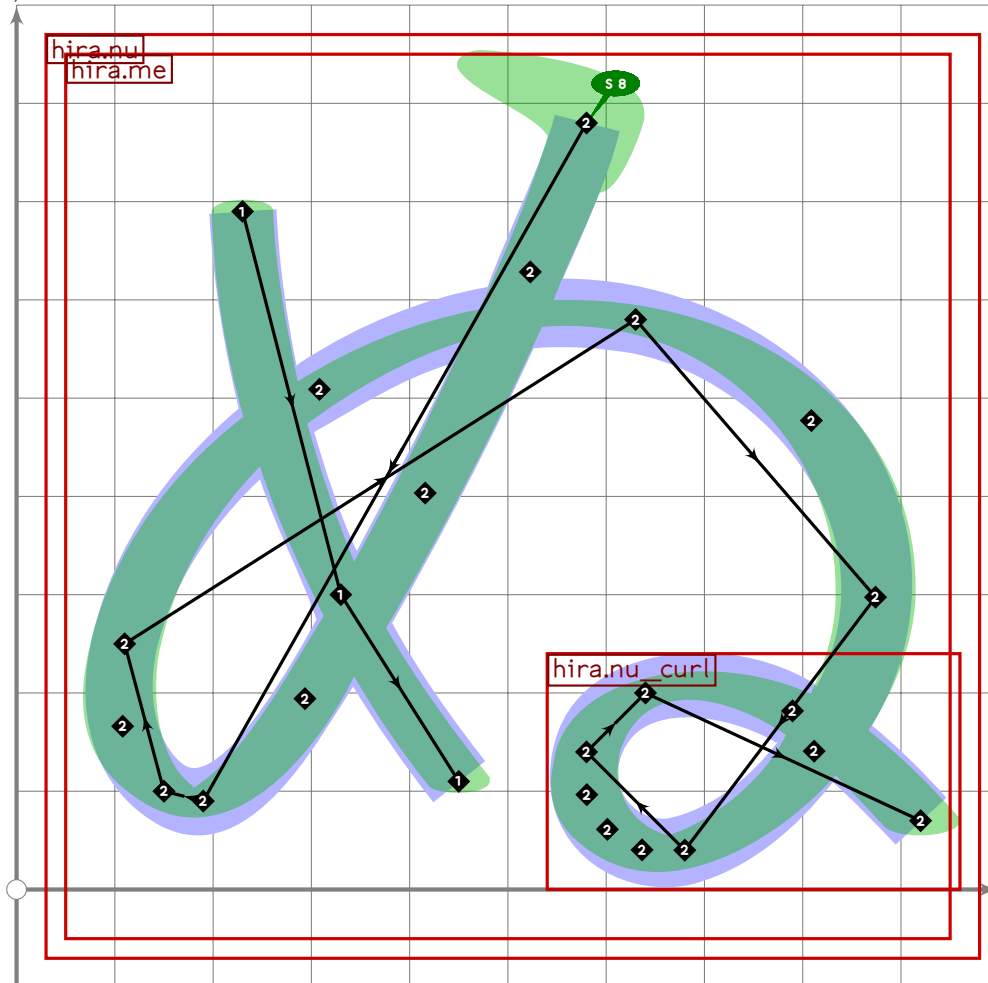
```



```

447 set_boserif(0,4,6);
448 expand_pbox;
449 enddef;
450
451 vardef hira.nu_curl =
452   begingroup
453     push_pbox_explicit("hira.nu_curl",
454       identity xyscaled (420,240) shifted (540,0));
455     (680,40)..(580,140)..(640,200)..{curl 0}(920,70)
456   endgroup
457 enddef;

```



```

458
459 vardef hira.nu =
460   push_pbox_toexpand("hira.nu");
461
462   hira.me;
463
464   replace_strokep(0)((subpath (0,4,8) of oldp)..tension 1.2..
465     hira.nu_curl);
466   replace_strokeq(0)((1,5,1,5)-(1,4,1,4)-(1,6,1,6)-(1,4,1,4)-
467     (1,6,1,6)-(1,6,1,6)-(1,7,1,7)-(1,3,1,3)-(1,6,1,6));

```

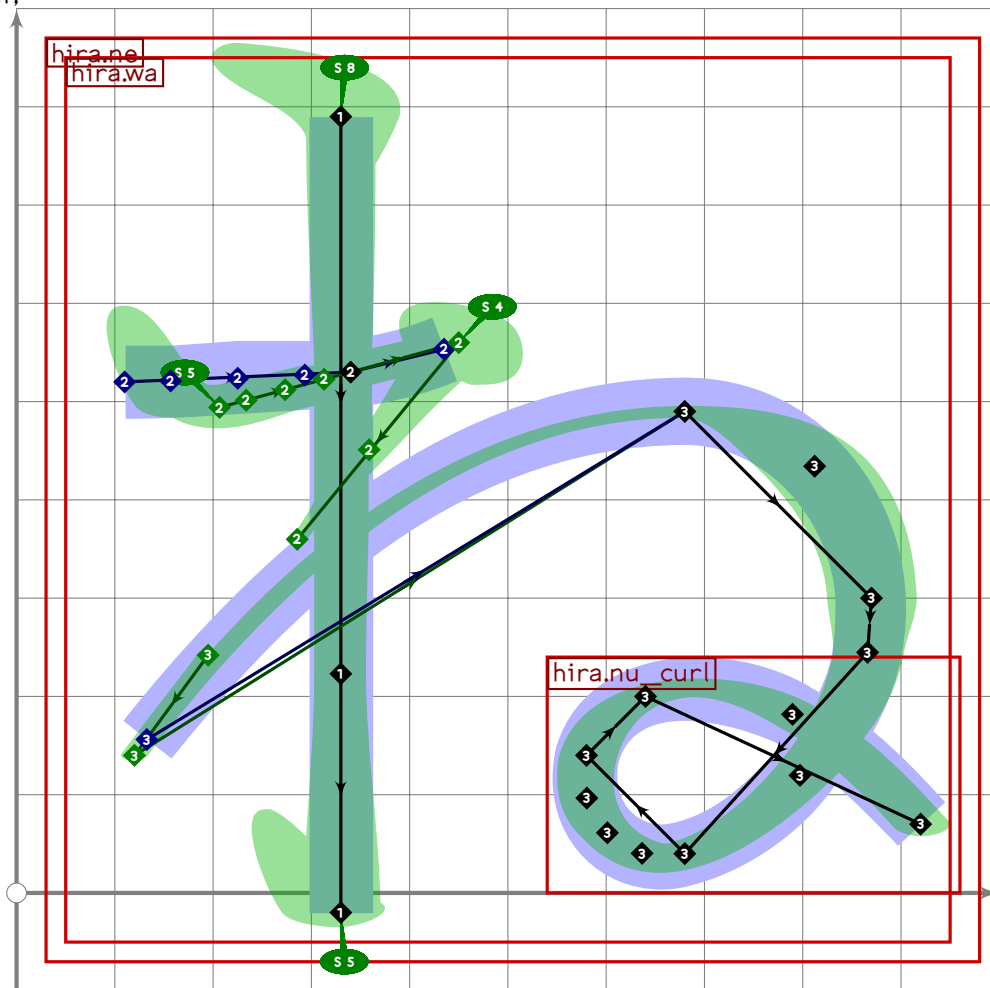
HIRA

U+306D
tsuku.uni306D

```

468 set_boserif(0,0,8);
469 expand_pbox;
470 enddef;

```

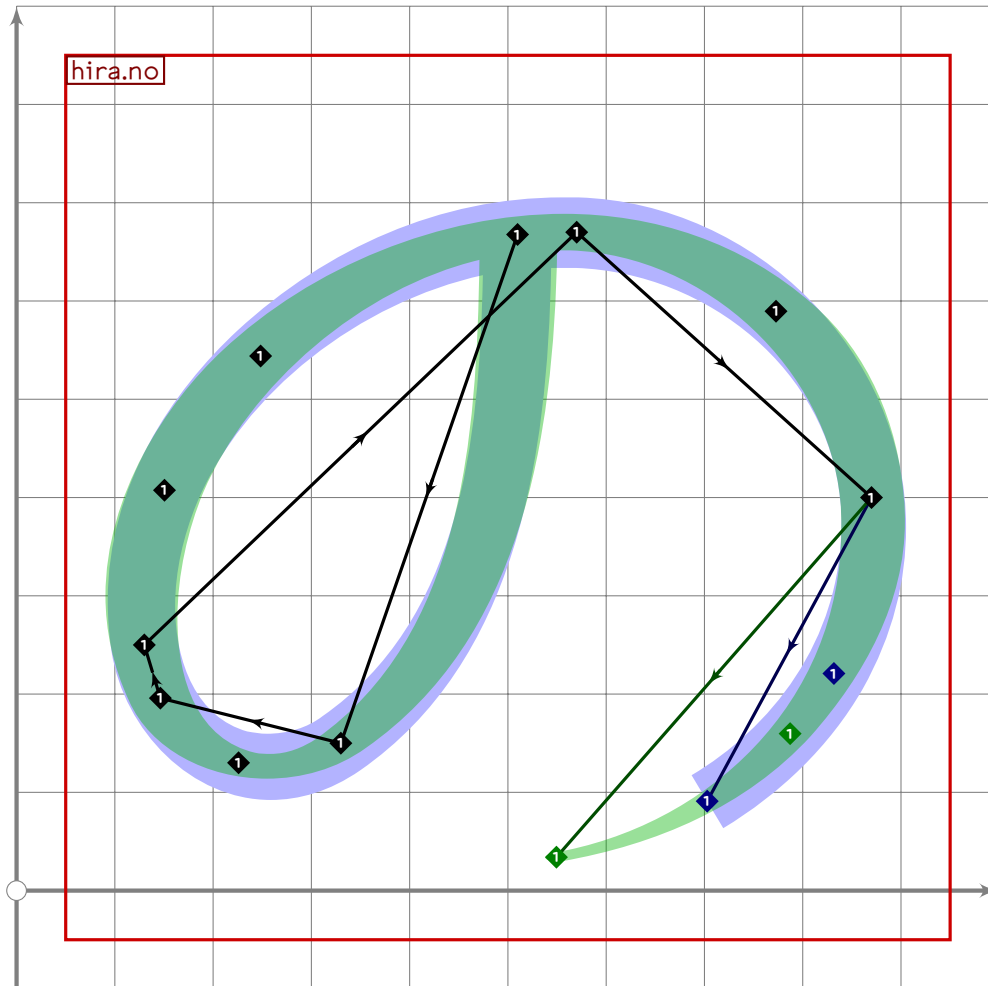


```

471
472 vardef hira.ne =
473   push_pbox_toexpand("hira.ne");
474
475   hira.wa;
476
477   replace_strokeq(0)((subpath (0,6,1) of oldp)..tension 1.2..
478     hira.nu_curl);
479   replace_strokeq(0)((2,2)-(1.6,1.6)-(2.2,0.9)-(0.7,0.7)-(0.97,0.97)-
480     (2,2)-(1.5,1.5)-(1.4,1.4)-(1.4,1.4)-(1.2,1.2)-(1.3,1.3));
481   expand_pbox;
482 enddef;

```

HIRA



```

483
484 vardef hira.no =
485   push_pbox__toexpand("hira.no");
486
487   begingroup
488     save px,py;
489     path px,py;
490     px:=(410,30)..(130,250)..tension 1.1..(570,670)..(870,400)..cycle;
491     py:=(510,770){down}{dir 215}(330,150);
492
493     px:=subpath (0.85,4) of px;
494     push_stroke(
495       (subpath (xpart (py intersectiontimes px),infinity) of py)..px,
496       (1.6,1.6)-(1.3,1.3)-(1.4,1.4)-(1.5,1.5)-(1.8,1.8)-
497       (1.4,1.4)-(0.7,0.7));
498     endgroup;
499     expand_pbox;
500 enddef;
501

```

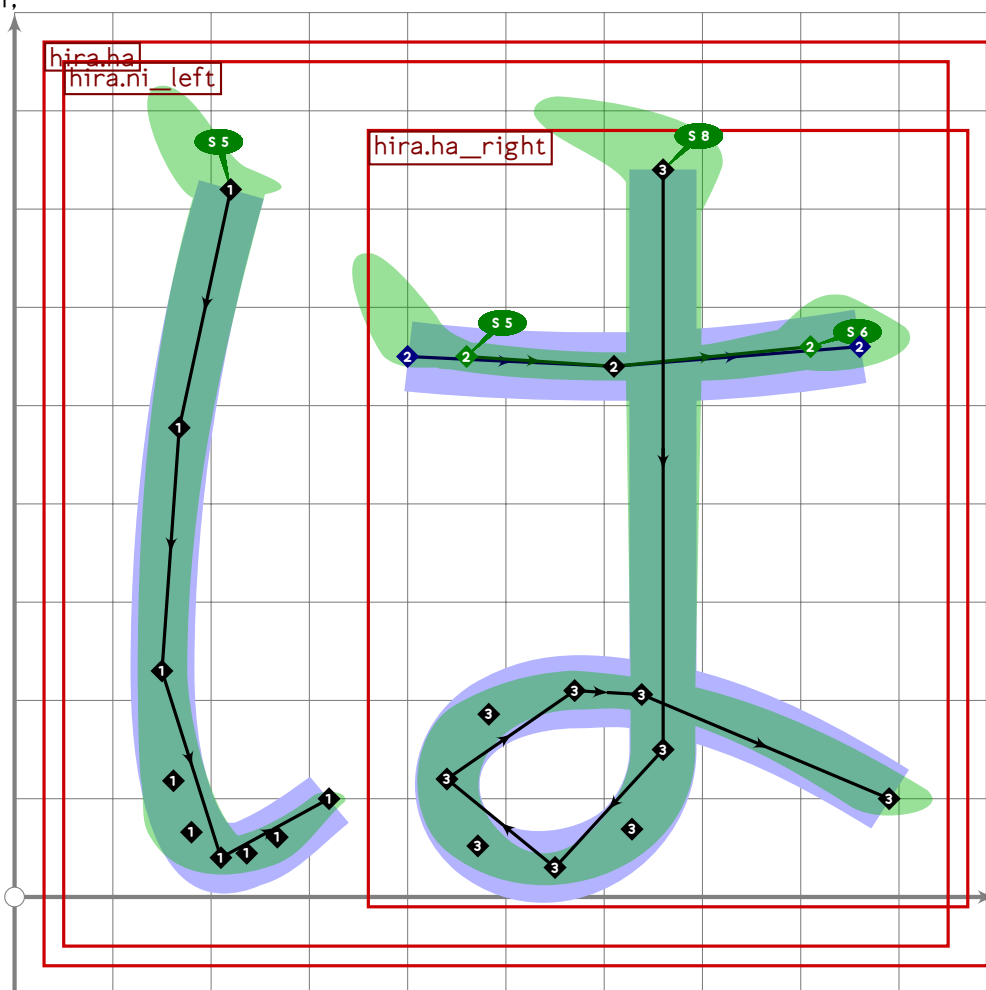
HIRA

Hiragana Hahifuheho/Babibubebo/Papipupepo

```

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

```



HIRA

```

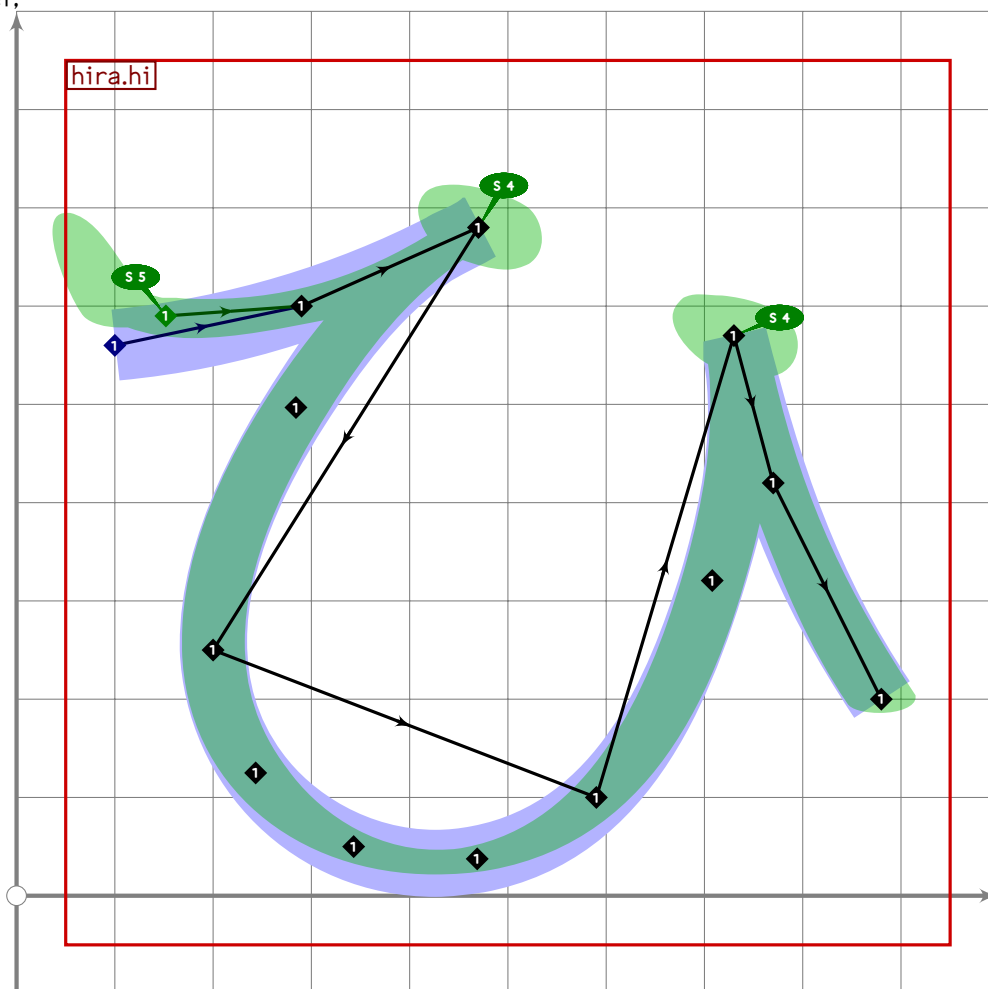
513
514 vardef hira.ha =
515   push_pbox_toexpand("hira.ha");
516
517   hira.ni_left;
518
519   push_stroke((400+60*mincho,550)..(610,540)..(860-50*mincho,560),
520     (1.6,1.6)..(1.6,1.6)..(2,2));

```

```

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

```



```

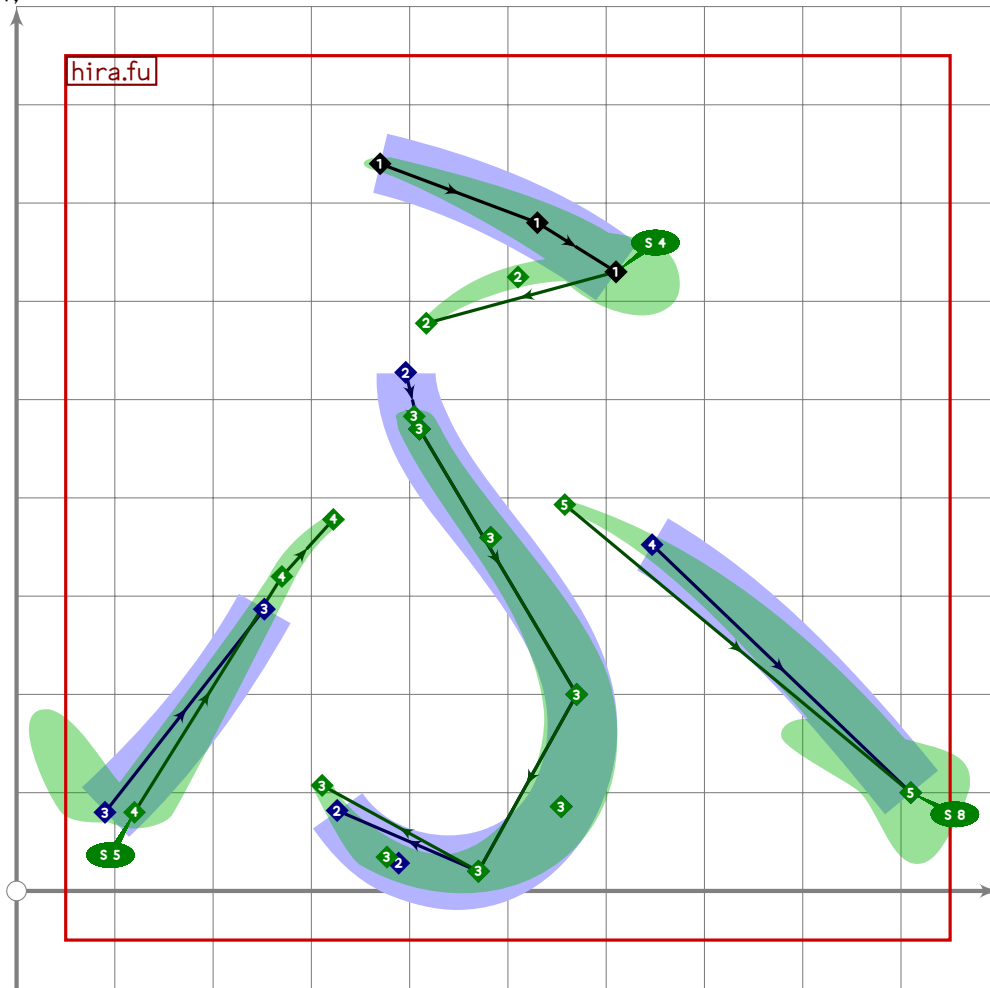
528
529 vardef hira.hi =
530   push_pbox_toexpand("hira.hi");
531
532   push_stroke(((100,560)+60*mincho*dir 30)..(290,600)..
533     {curl 1}(470,680){curl 1}..
534     tension 1.3..(200,250)..(590,100)..tension 1.3..
535     {curl 1}(730,570){curl 1}..(770,420)..(880,200),
536     (1,8,1,8)-(1,7,1,7)-(1,5,1,5)-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

```

HIRA

U+3075
tsuku.uni3075

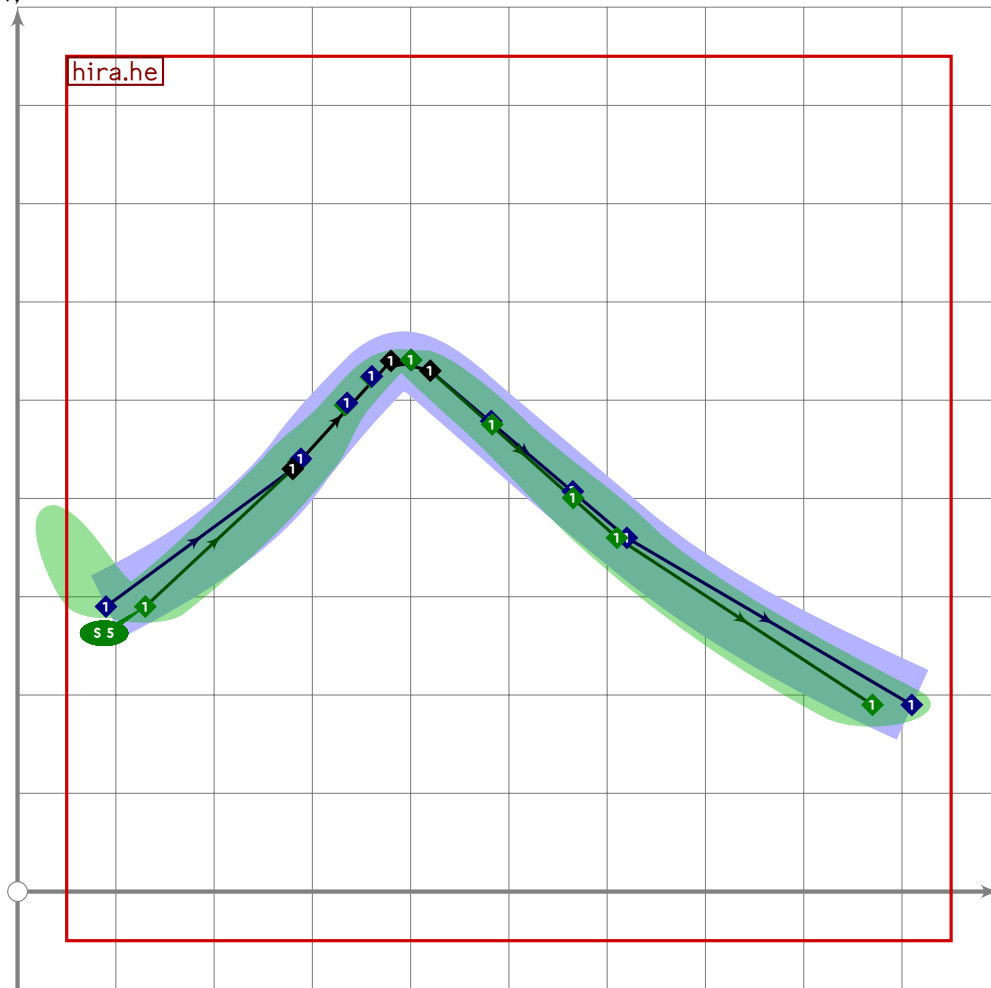
```
542 set_boserif(0,5,4);  
543 expand_pbox;  
544 enddef;
```



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

HIRA

563 enddef;



564

565 vardef hira.he =

566 push_pbox_toexpand("hira.he");

567

568 push_stroke((90+40*mincho,290){curl 0.2}..(280,430)..tension 2..

569 (380,540)..(420,530)..tension 2..(620-10*mincho,360)..

570 {curl 0.2}(910-40*mincho,190),

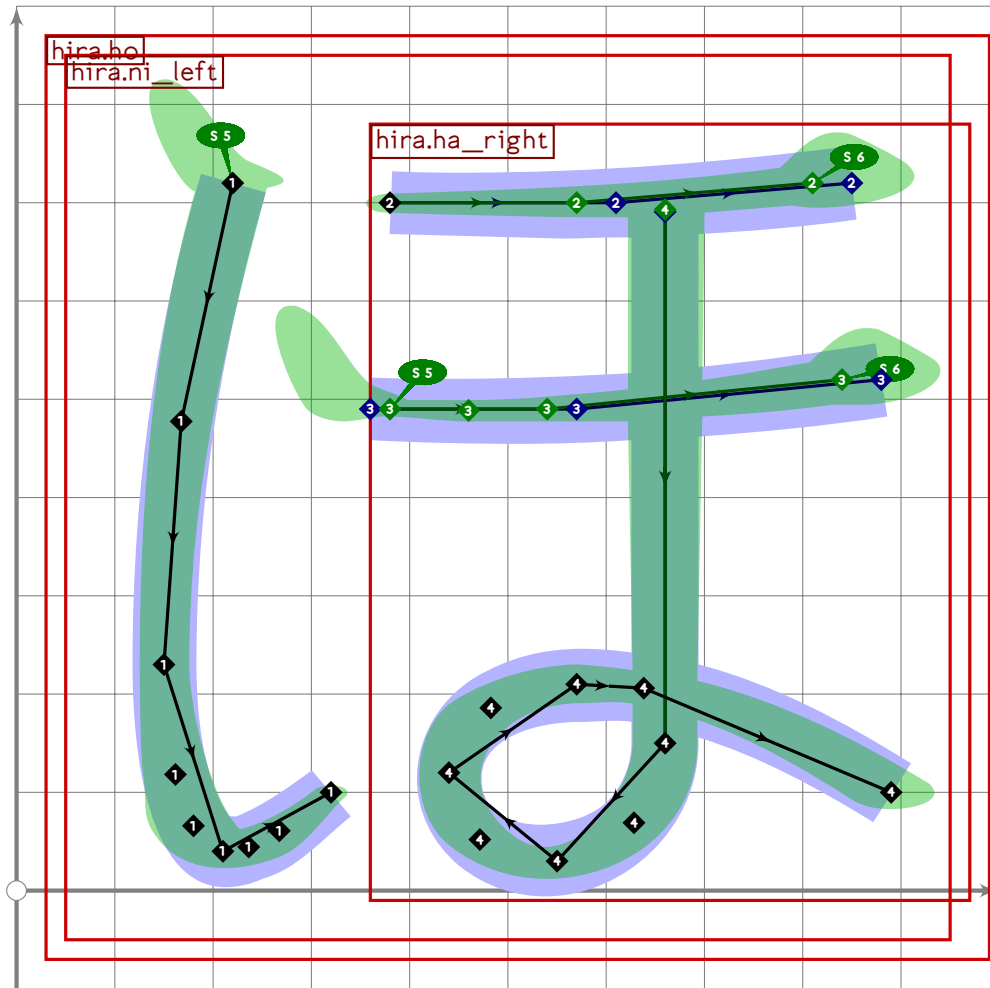
571 (1.7,1.7)..(1.6,1.6)..(1.2,1.2)..(1.3,1.3)..(1.7,1.7)..(2.1,2.1));

572 set_boserif(0,0,5);

573 expand_pbox;

574 enddef;

HIRA



```

575
576 vardef hira.ho =
577   push_pbox_toexpand("hira.ho");
578
579   hira.ni_left;
580
581   push_stroke((380,700)..(610-40*mincho,700)..(850-40*mincho,720),
582     (1,2,1,2)..(1,6,1,6)..(1,9,1,9));
583   set_boserif(0,2,6);
584
585   push_stroke((360+20*mincho,490)..(570-30*mincho,490)..(880-40*mincho,520),
586     (1,2,1,2)..(1,5,1,5)..(2,2));
587   set_boserif(0,0,5);
588   set_boserif(0,2,6);
589
590   hira.ha_right;
591   replace_strokep(0)(subpath
592     (xpart (oldp intersectiontimes get_strokep(-2))+0.02,infinity) of oldp);
593   expand_pbox;
594 enddef;
595

```

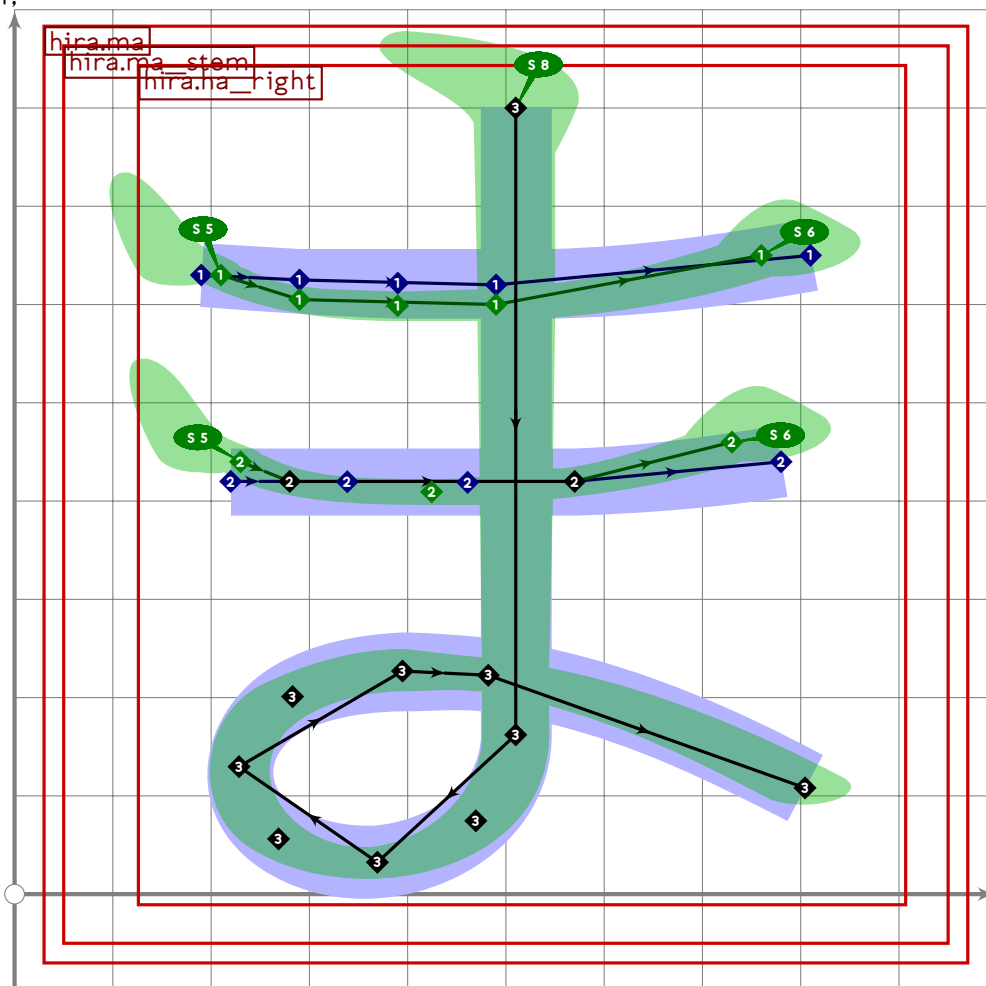
HIRA

Hiragana Mamimumemo

```

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

```



HIRA

```

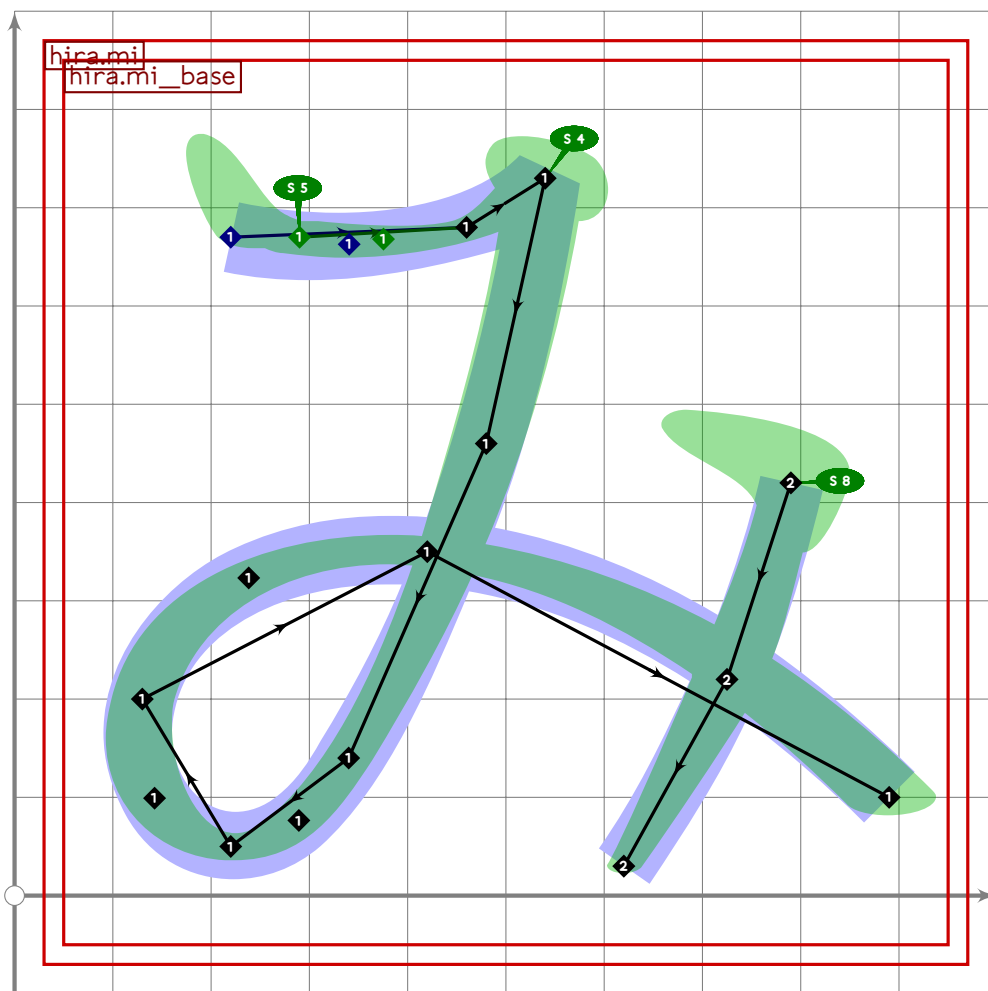
613
614 vardef hira.ma =

```

```

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

```

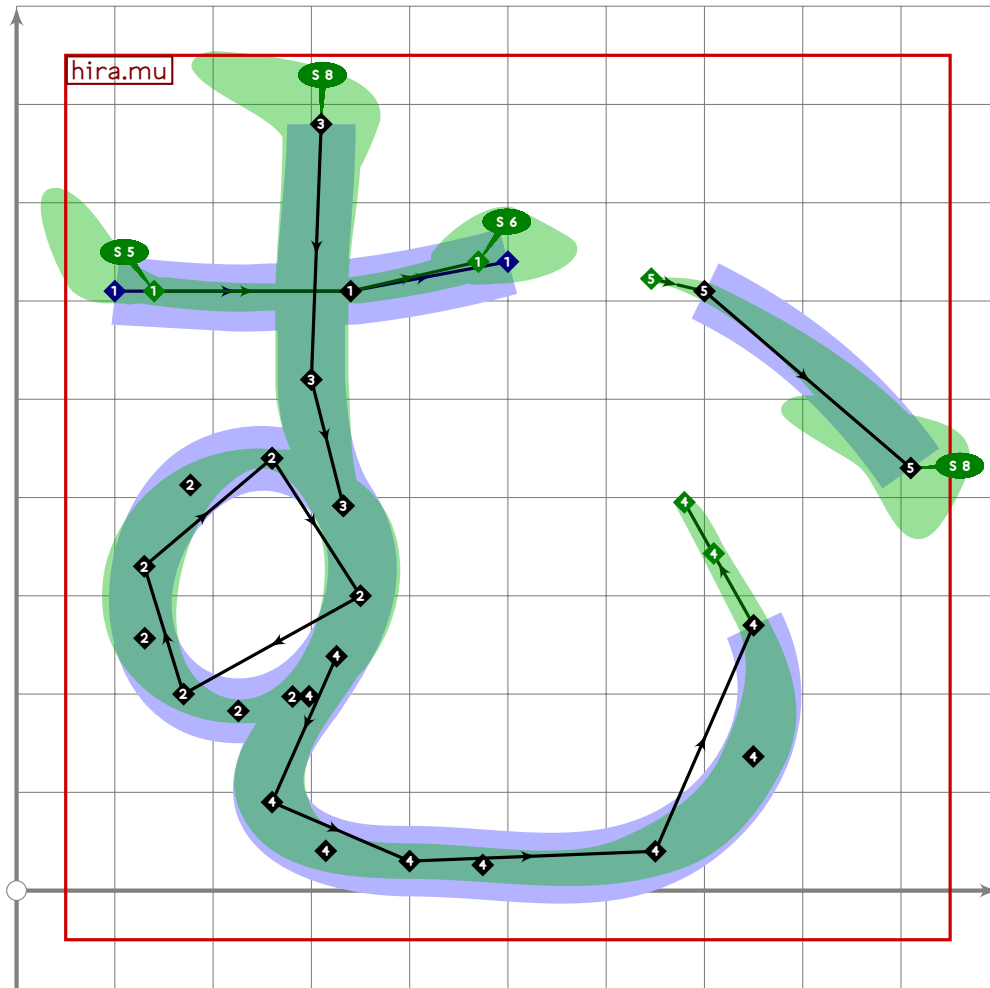


```

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

```

HIRA



```

660
661 vardef hira.mu =
662   push_pbox_toexpand("hira.mu");
663
664   push_stroke((100+40*mincho,610)..(340,610)..(500-30*mincho,640),
665     (1,6,1,6)-(1,4,1,4)-(1,5,1,5));
666   set_boserif(0,0,5);
667   set_boserif(0,2,6);
668
669   push_stroke((260,440)..(350,300)..(170,200)..(130,330)..cycle,
670     (1,3,1,3)..(1,6,1,6)..(1,3,1,3)..(1,6,1,6)..cycle);
671
672   push_stroke((310,780){down}..(300,520)..
673     (point 0.5 of get_stroke(0)){direction 0.5 of get_stroke(0)},
674     (1,6,1,6)-(1,5,1,5)-(1,2,1,2));
675   set_boserif(0,0,8);
676
677   push_stroke(
678     (point 1.25 of get_stroke(-1)){direction 1.20 of get_stroke(-1)}..
679     (260,90)..(400,30){right}..(650,40)..(750,270)..
680     tension 2..(600,590)..(700,610)..(910,430),

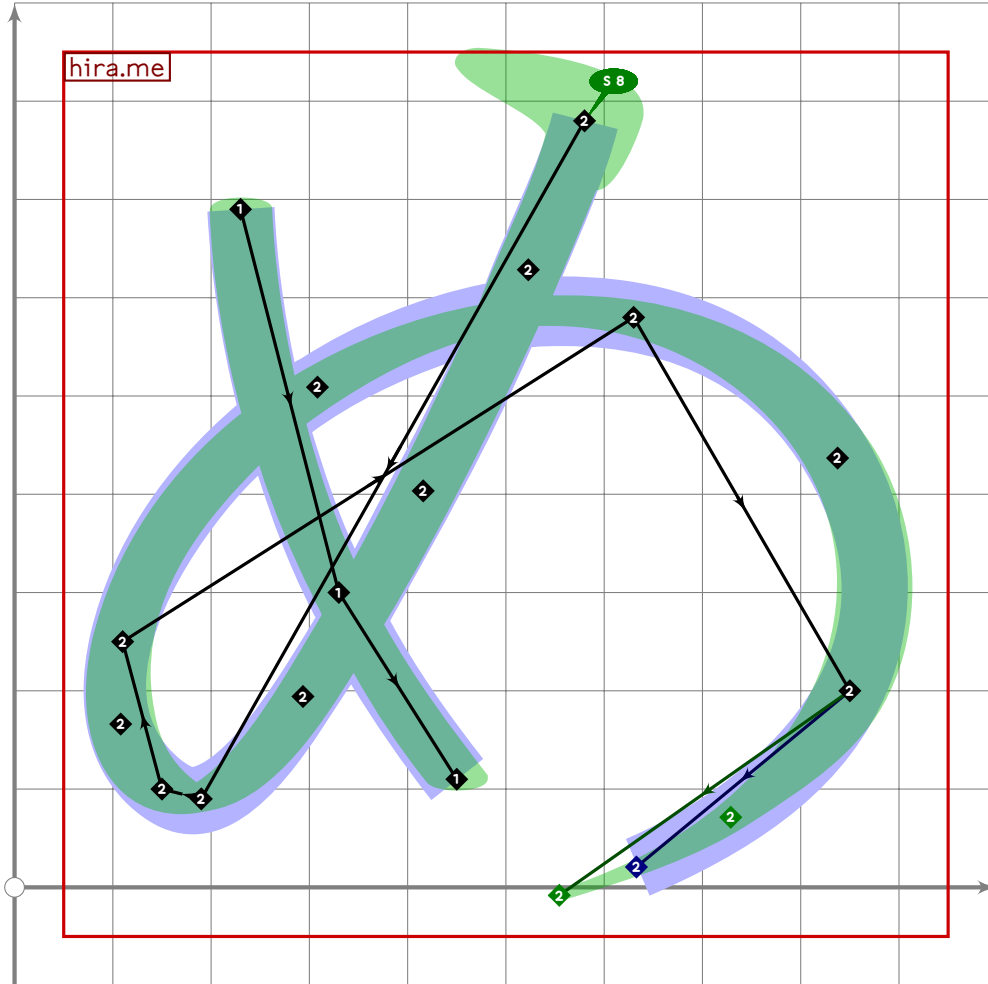
```

HIRA

```

681 (14,14)-(1.5,1.5)-(1.8,1.8)-(1.9,1.9)-
682 (1,1)-(0.6,0.6)-(1,1)-(1.6,1.6));
683 set_boserif(0,78);
684 expand_pbox;
685 enddef;

```

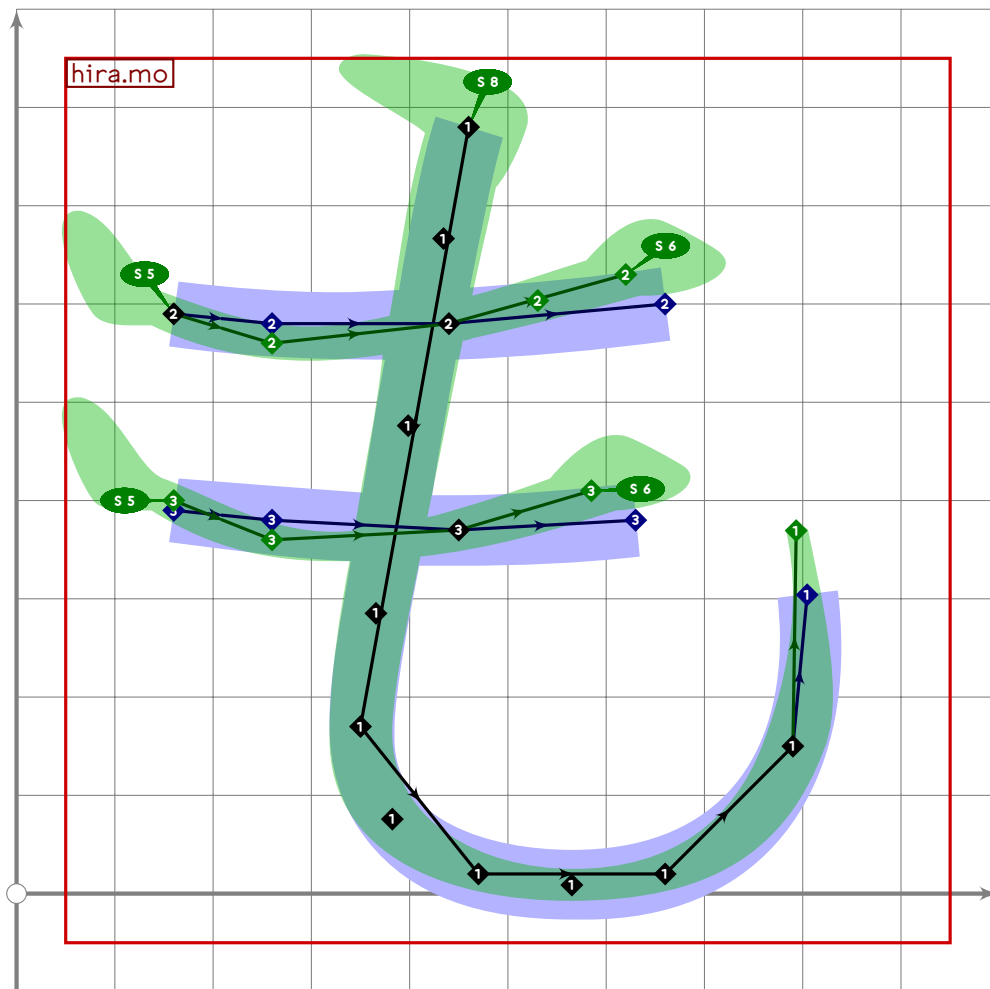


```

686
687 vardef hira.me =
688   push_pbox_toexpand("hira.me");
689
690   push_stroke((230,690)..(330,300)..(450,110),
691     (14,14)-(1.3,1.3)-(14,14));
692
693   push_stroke((580,780){curl 0.2}..tension 2.5..(190,90)..
694     (150,100)..(110,250)..tension 1.1..(630,580)..
695     (850,200)..tension 1.1..{curl 0.2}(470,30),
696     (1.5,1.5)-(14,14)-(1.6,1.6)-(14,14)-
697     (1.6,1.6)-(1.6,1.6)-(0.65,0.65));
698   set_boserif(0,0,8);
699   expand_pbox;
700 enddef;

```

HIRA



```

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

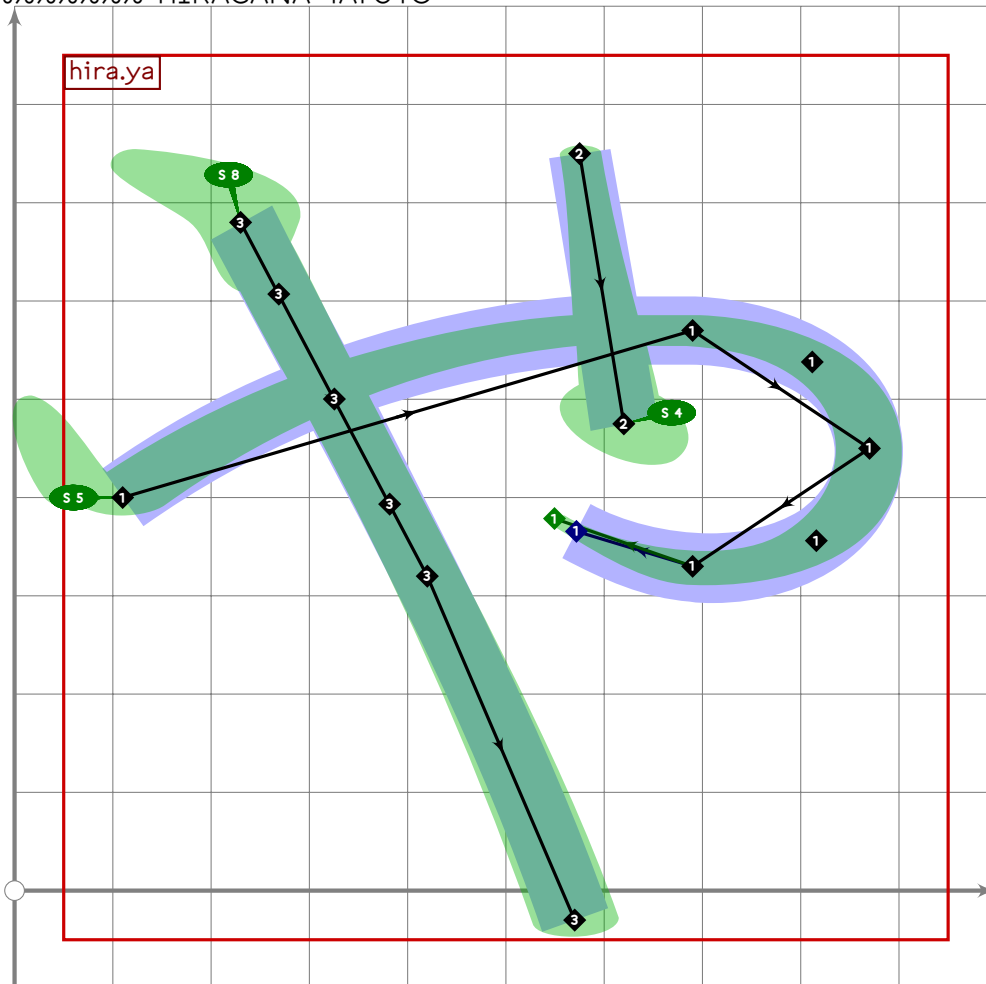
```

HIRA

722 enddef;
723

Hiragana Yayuyo

724 %%%%%%%%% HIRAGANA YAYUYO



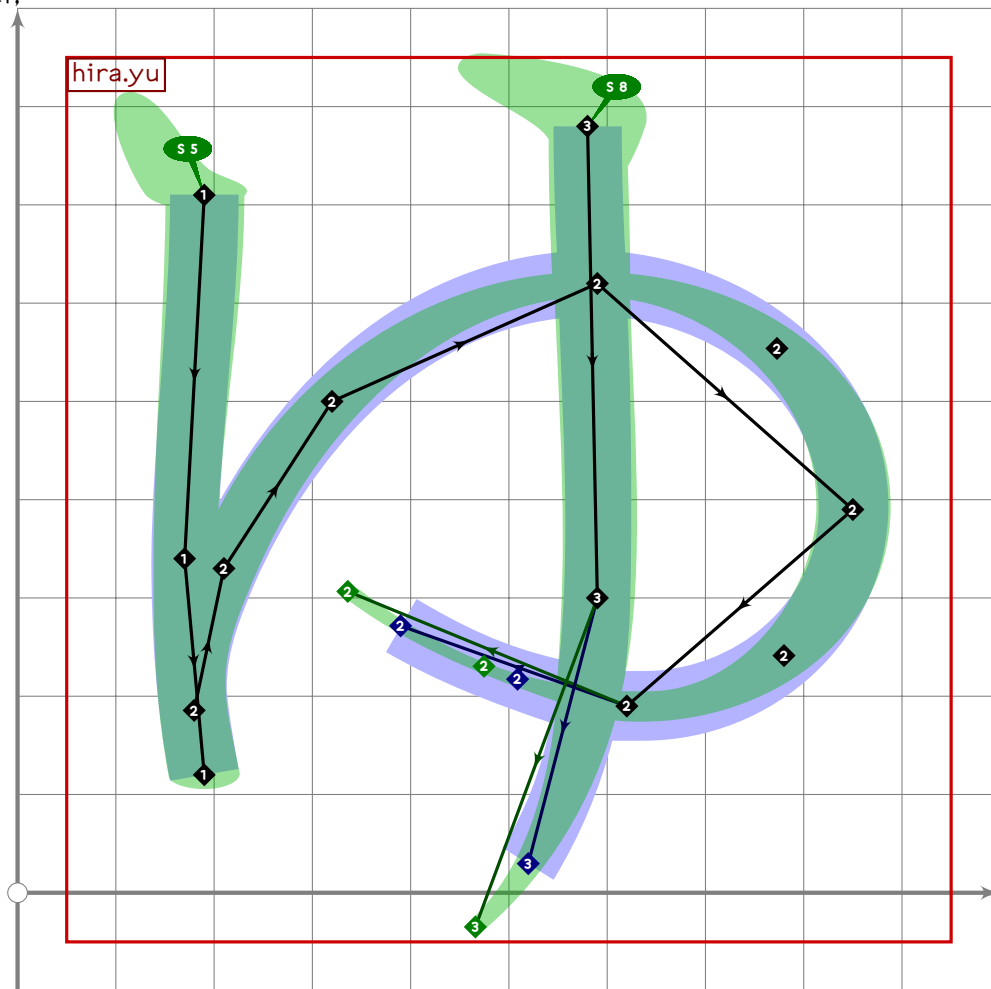
```

725
726 vardef hira.ya =
727   push_pbox_toexpand("hira.ya");
728
729   push_stroke((110,400)..(690,570)..(870,450)..(690,330)..(520,400),
730     (1.8,1.8)-(1.6,1.6)-(1.4,1.4)-(1.8,1.8)-(0.6,0.6));
731   set_boserif(0,0,5);
732
733   push_stroke((575,750)-(620,475),
734     (1.1,1.1)-(1.6,1.6));
735   set_boserif(0,1,4);
736
737   push_stroke((230,680)..tension 2..(420,320)..(570,-30),
738     (1.6,1.6)-(1.4,1.4)-(1.7,1.7));
739   set_boserif(0,0,8);

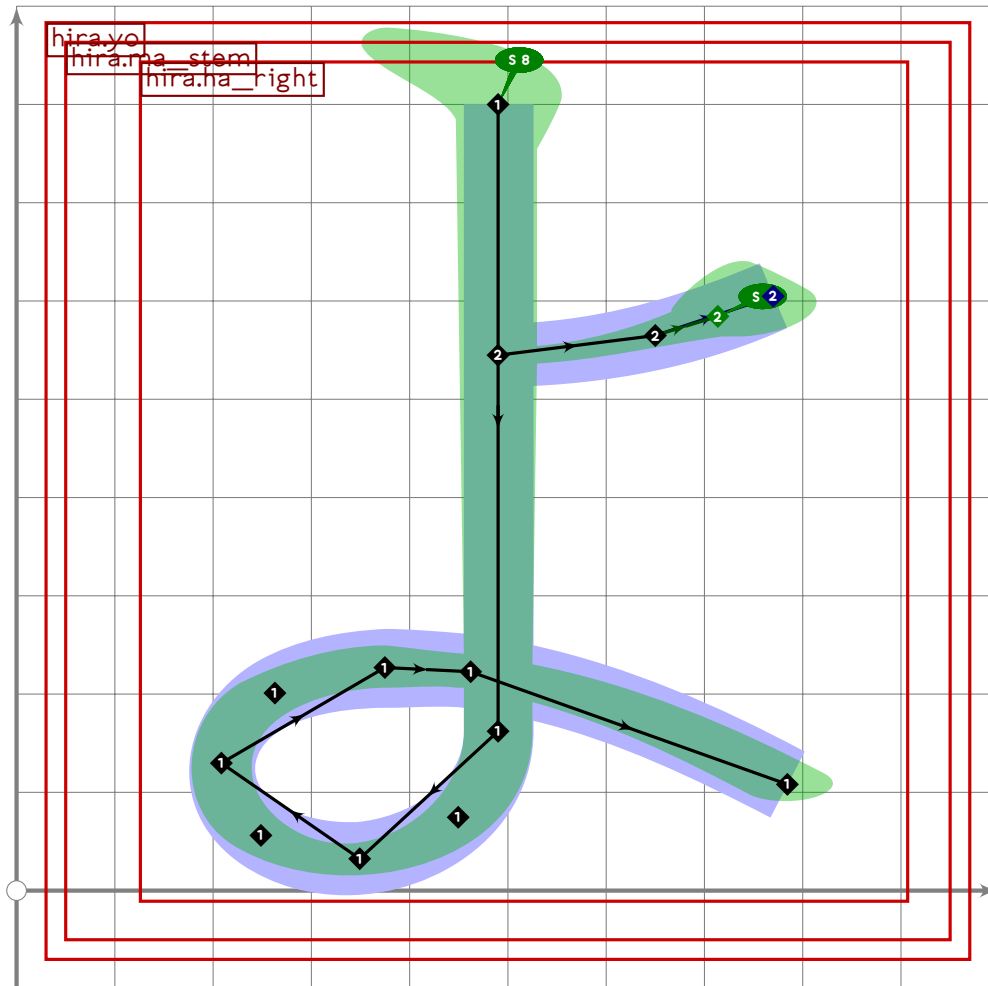
```

HIRA

```
740 expand_pbox;
741 enddef;
```



```
742
743 vardef hira.yu =
744   push_pbox_toexpand("hira.yu");
745
746   push_stroke((190,710){down}...(170,340)..(190,120),
747     (1.6,1.6)–(1.4,1.4)–(1.5,1.5));
748   set_boserif(0,0,5);
749
750   push_stroke((point 1.7 of get_stroke(0))
751     {–direction 1.7 of get_stroke(0)}..(210,330)..
752     (320,500)..(590,620)..(850,390)..(620,190)..tension 1.3..(320,320),
753     (1.4,1.4)–(1.5,1.5)–(1.4,1.4)–(1.5,1.5)–(1.6,1.6)–
754     (1.6,1.6)–(0.77,0.77));
755
756   push_stroke((580,780){down}...(590,300)..{dir 190}(360,85),
757     (1.6,1.6)–(1.5,1.5)–(0.6,0.6));
758   set_boserif(0,0,8);
759   expand_pbox;
760 enddef;
```

```

761
762 vardef hira.yo =
763   push_pbox_toexpand("hira.yo");
764
765   hira.ma_stem;
766
767   replace_strokep(0)(oldp shifted (-20,0));
768   z0=point 0.4 of get_strokep(0);
769
770   push_stroke(z0..(z0+(160,20))..(z0+(280,60)+60*mincho*dir 200),
771     (1.2,1.2)-(1.4,1.4)-(1.8,1.8));
772   set_boserif(0,2,6);
773   expand_pbox;
774 enddef;
775

```

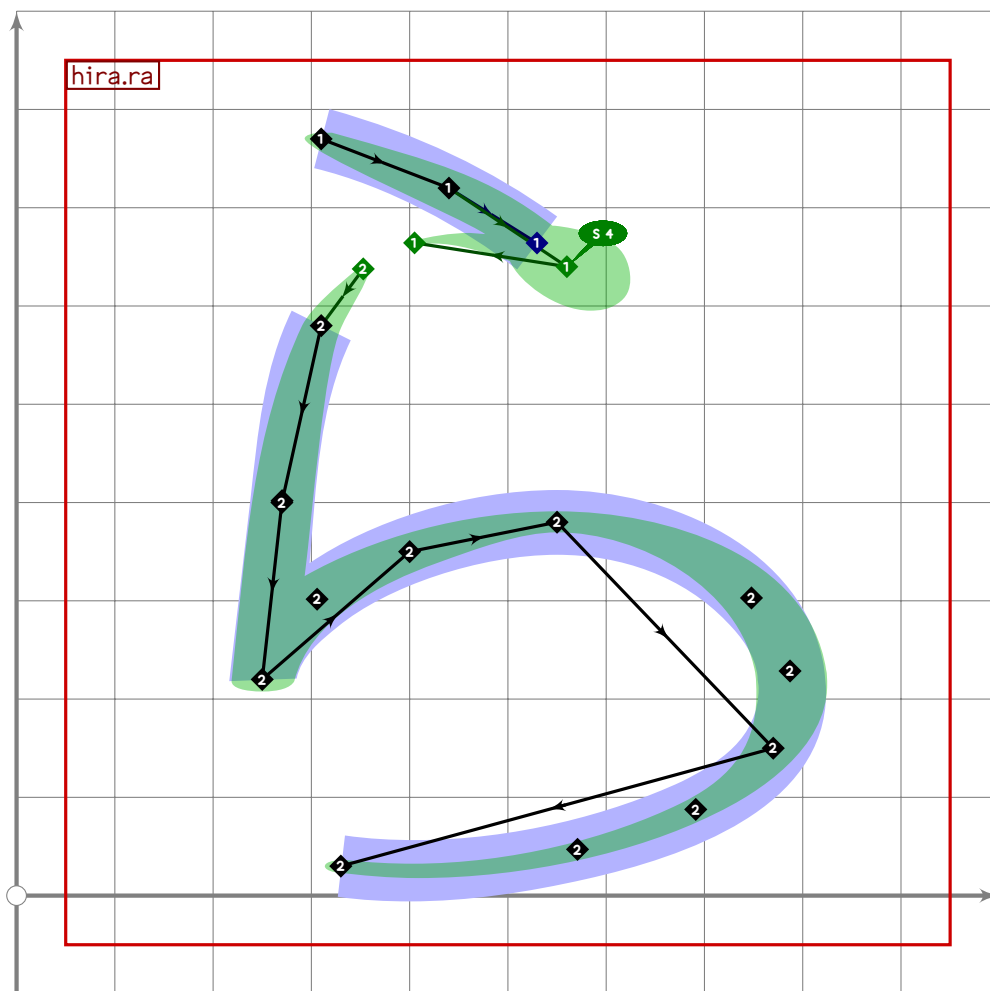
HIRA

Hiragana Rarirurero

```

776 %%%%%%%%%% HIRAGANA RARIRURERO

```

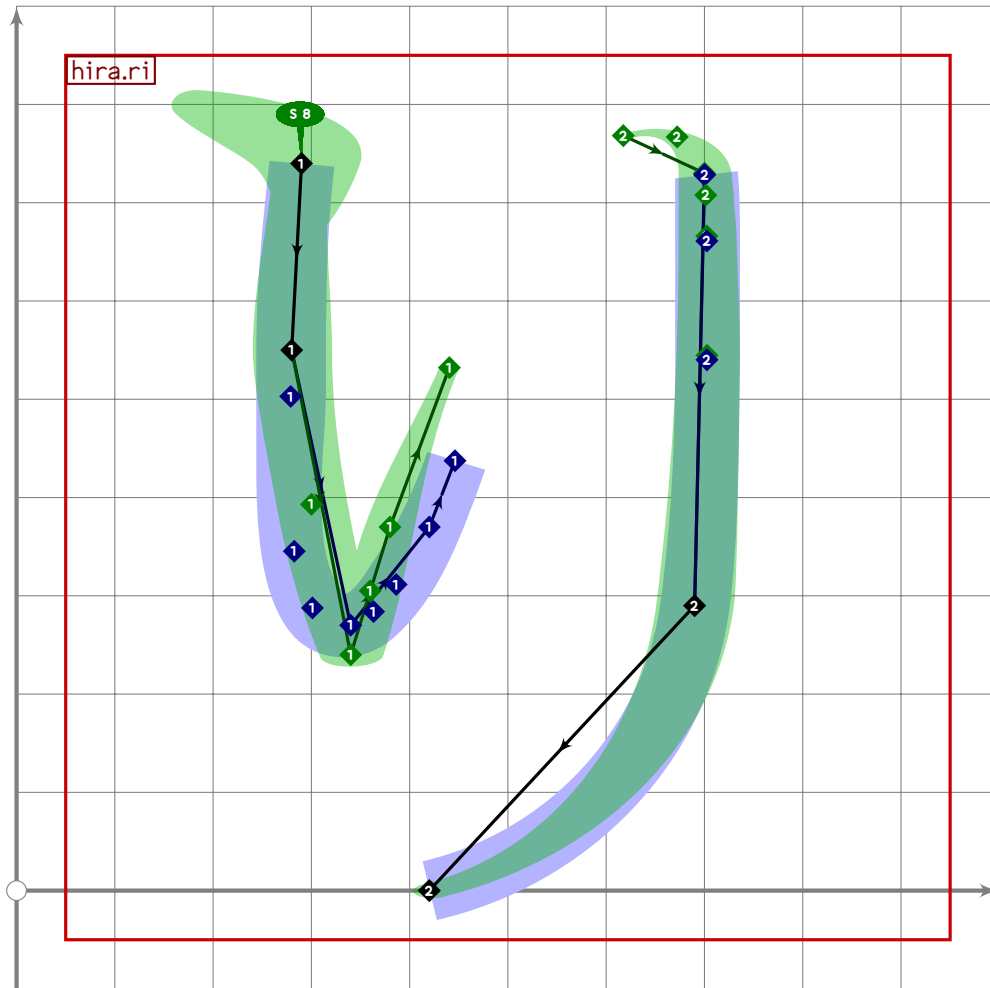


```

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

```

HIRA



```

793
794 vardef hira.ri =
795   push_pbox__toexpand("hira.ri");
796
797   begingroup
798     save ripx,ripy,ripz,x,y;
799     path ripx,ripy,ripz;
800     numeric x[],y[];
801     z1=(290,740);
802     z2=(280,550);
803     z3=(340,270-30*mincho);
804     z4=(420-40*mincho,370);
805     z5=(540,710);
806     z6=(700,730);
807     ripx=z1..z2{down}..tension 1.5..z3..
808       tension 1.5..z4..z5..z6..tension 5 and 1.2..
809       (690,290)..tension 0.75 and 1.{curl 0.45}(420,0);
810     ripy=z1..z2{down}..tension 1.5..{curl 1}z3{curl 1}..
811       tension 1.5..z4..z5..z6..tension 5 and 1.2..
812       (690,290)..tension 0.75 and 1.{curl 0.45}(420,0);
813     push_stroke(interpath(mincho,ripx,ripy),

```

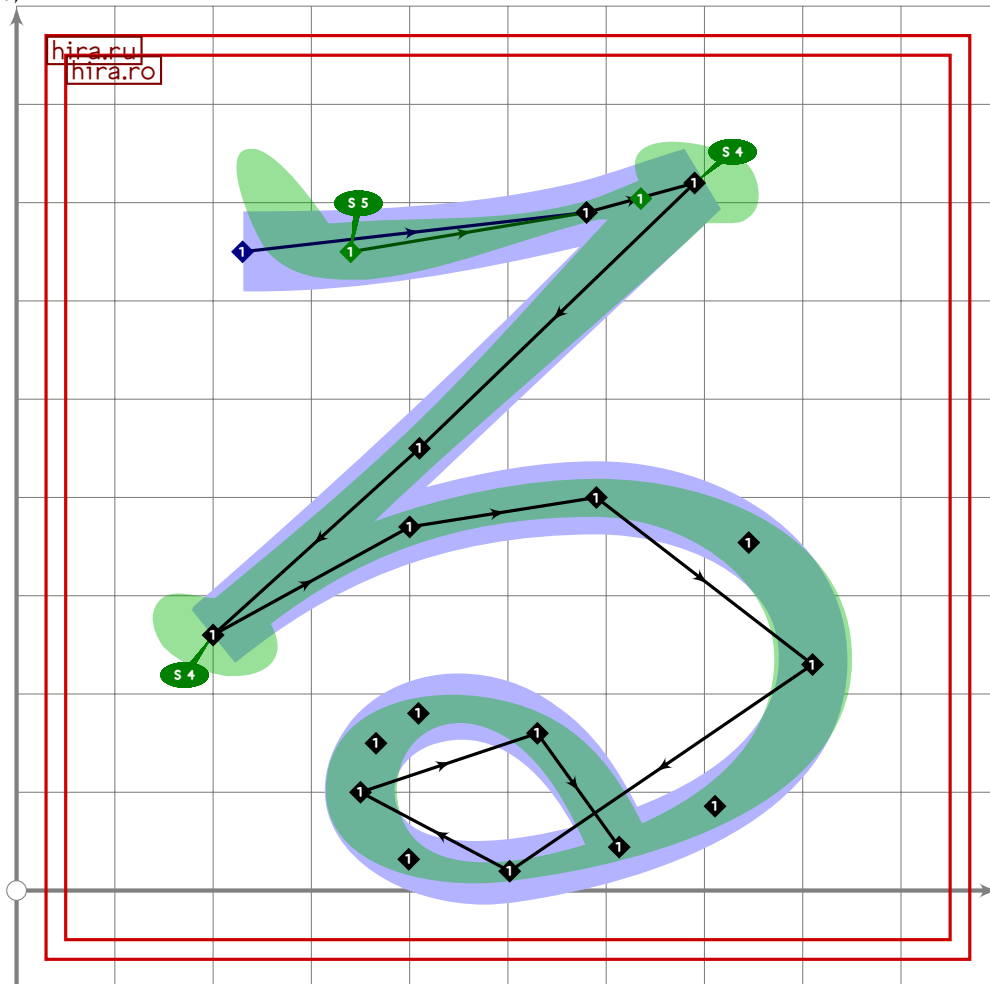
HIRA

U+308B
tsuku.uni308B

```

814      (1.3,1.3)–(1.6,1.6)–(14,14)–(1.2,1.2)–(04,0.2)–
815      (1.5,0.99)–(1.6,1.6)–(1,1));
816      set_boserif(0,0,8);
817      endgroup;
818      expand_pbox;
819  enddef;

```

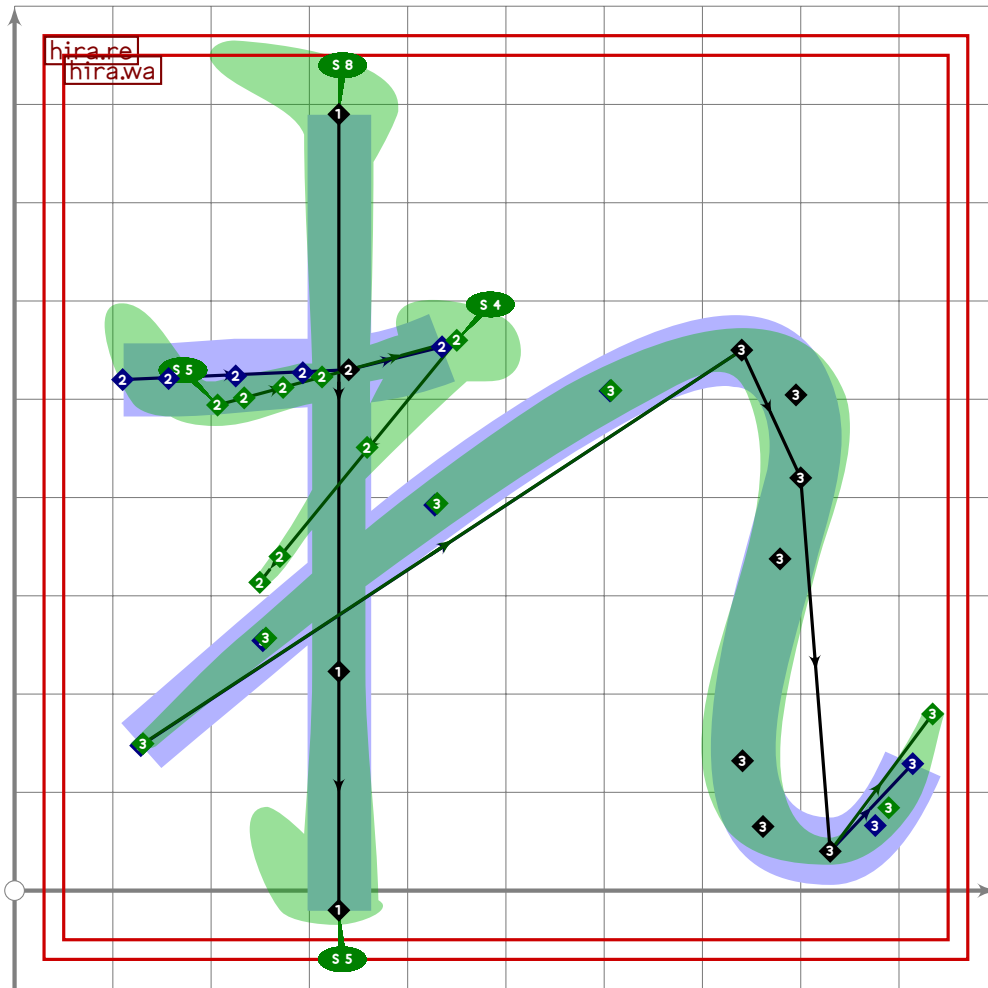


```

820
821  vardef hira.ru =
822    push_pbox_toexpand("hira.ru");
823
824    hira.ro;
825
826    replace_strokep(0)((subpath (0,78) of oldp)..(350,100)..(530,160)..
827      {curl 0.2}(point 7.6 of oldp));
828    replace_strokeq(0)((2.6,2.6)–(1.2,1.2)–(19,19)–
829      (1.3,1.3)–(1.6,1.6)–
830      (1.5,1.5)–(19,19)–(1.6,1.6)–
831      (1.2,1.2)–(1.5,1.5)–(14,14));
832    expand_pbox;
833  enddef;

```

HIRA

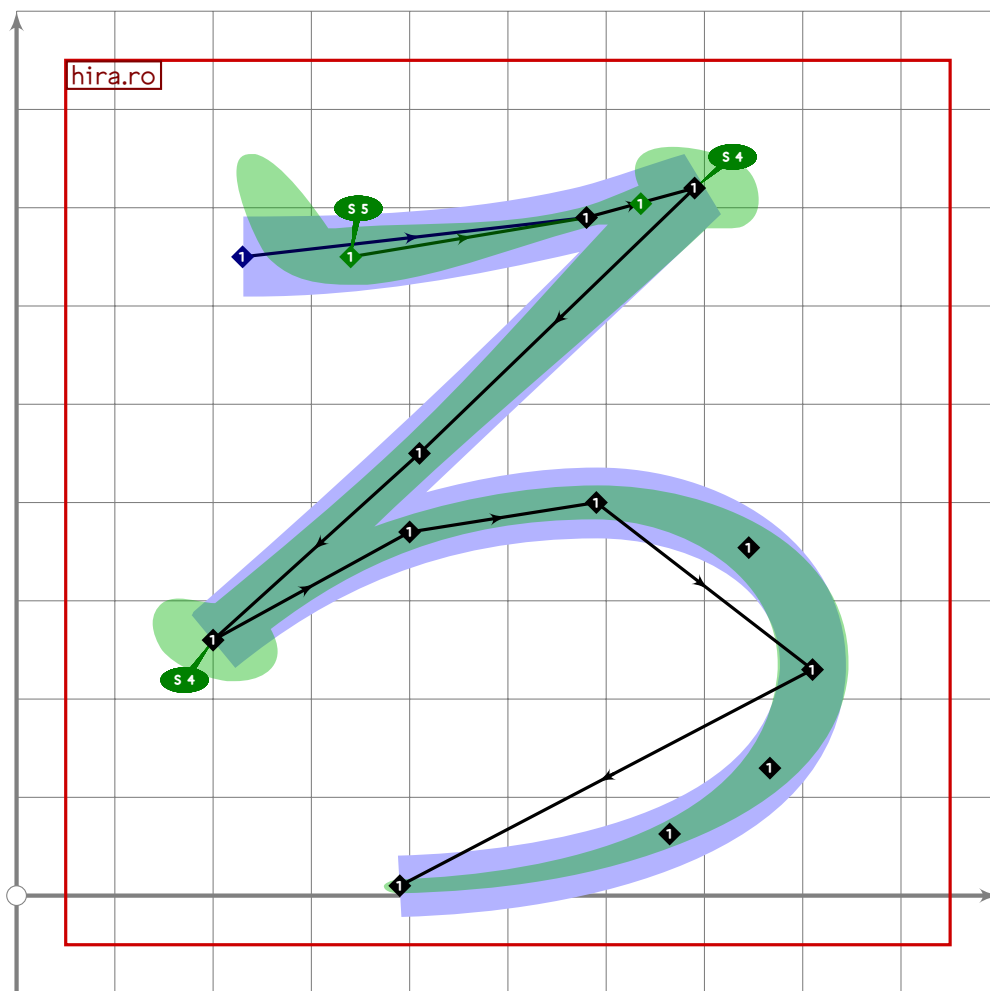


```

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

```

HIRA



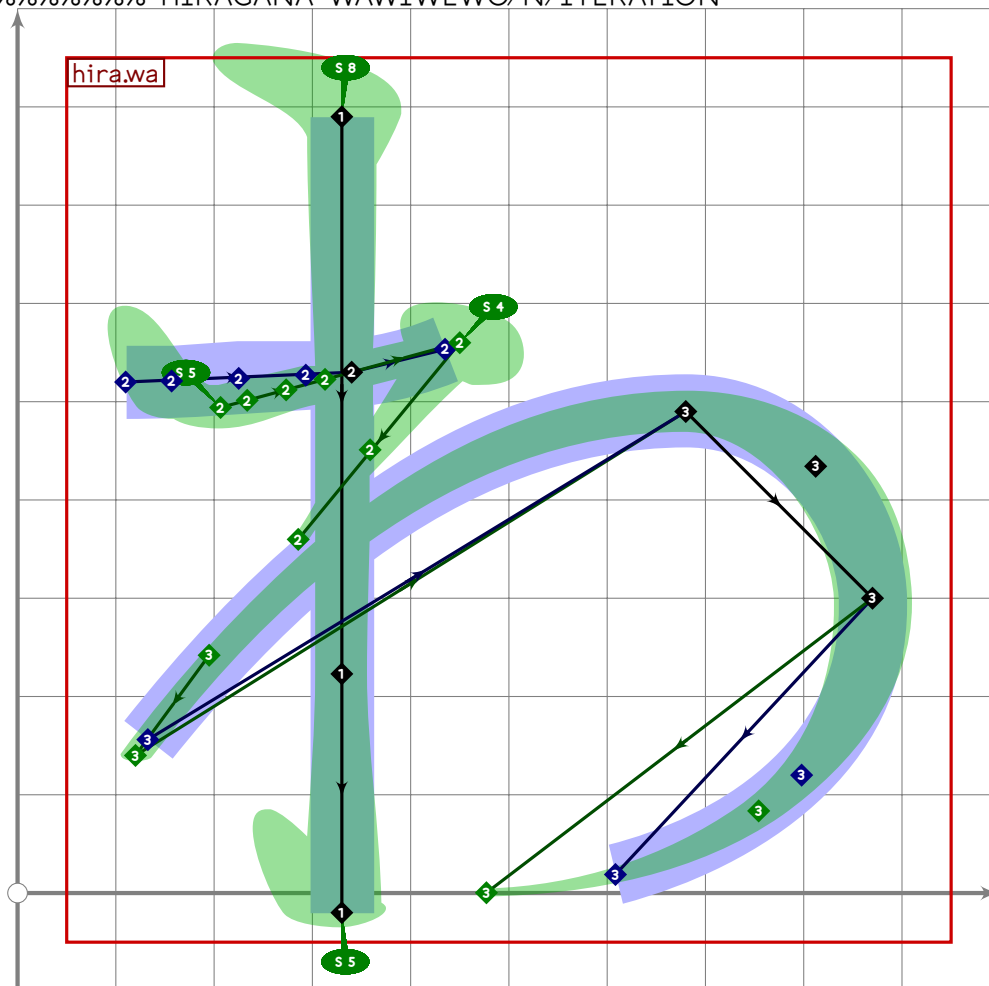
```

849
850 vardef hira.ro =
851   push_pbox_toexpand("hira.ro");
852
853   push_stroke((230+110*mincho,650)..(580,690)..{curl 1}{690,720}{curl 1}..
854     (410,450)..{curl 1}{200,260}{curl 1}..
855     (400,370)..(590,400){right}..(810,230)..
856     tension 1.1..{curl 0}{390,10},
857     (2.6,2.6)-(1.2,1.2)-(1.9,1.9)-
858     (1.3,1.3)-(1.6,1.6)-
859     (1.5,1.5)-(1.7,1.7)-(1.5,1.5)-(1,1));
860   set_botip(0,2,0);
861   set_botip(0,4,0);
862   set_boserif(0,0,5);
863   set_boserif(0,2,4);
864   set_boserif(0,4,4);
865   expand_pbox;
866 enddef;
867

```

Hiragana Wawiwewo/N/Iteration

868 %%%%%%%%% HIRAGANA WAWIWEWO/N/ITERATION



```

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

```

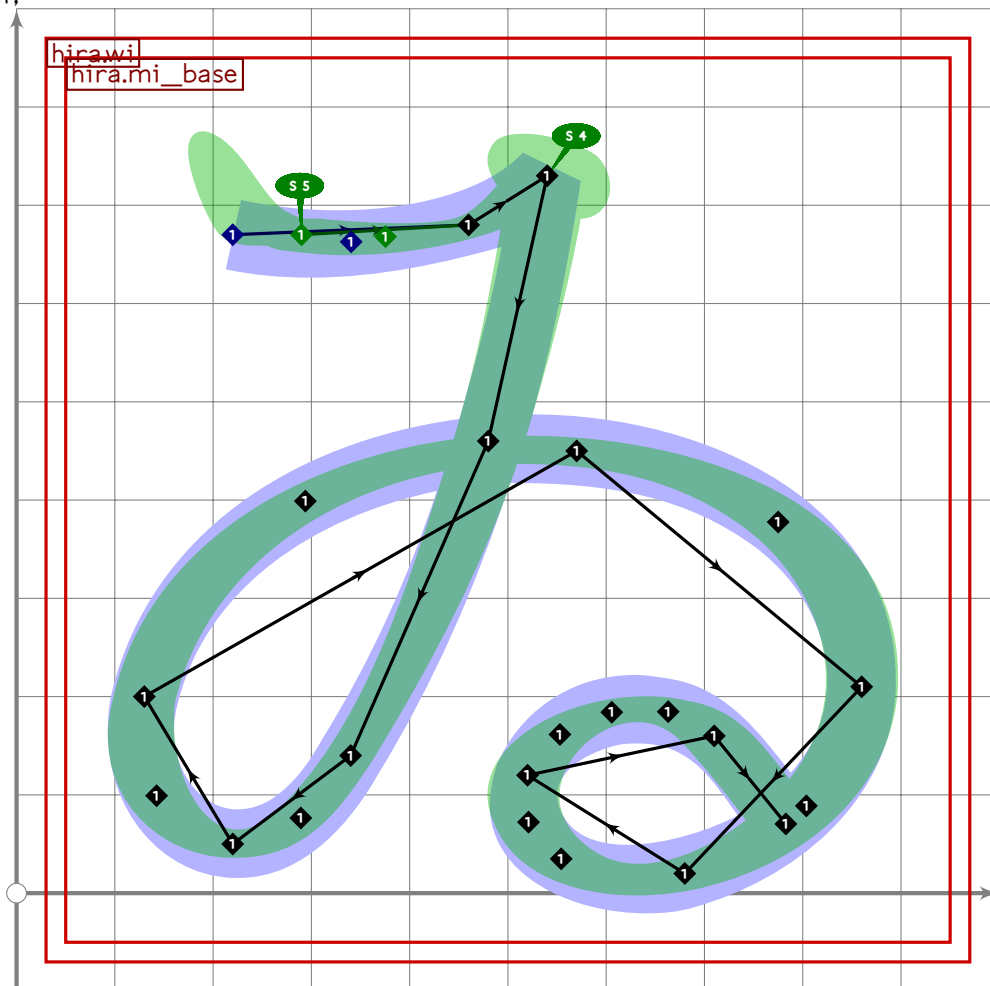
HIRA

U+3090
tsuku.uni3090

```

887 set_boserif(0,0,5);
888 set_boserif(0,2,4);
889 expand_pbox;
890 enddef;

```

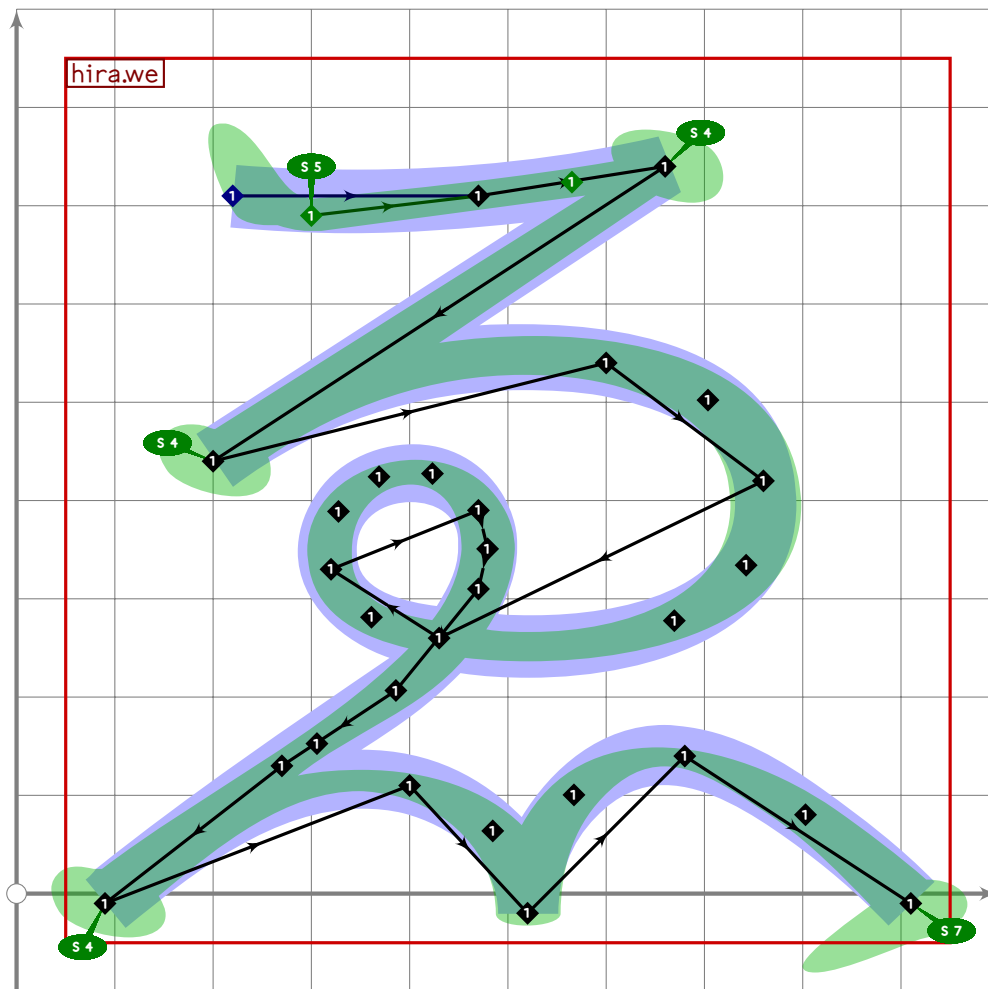


```

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

```

HIRA



```

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

```

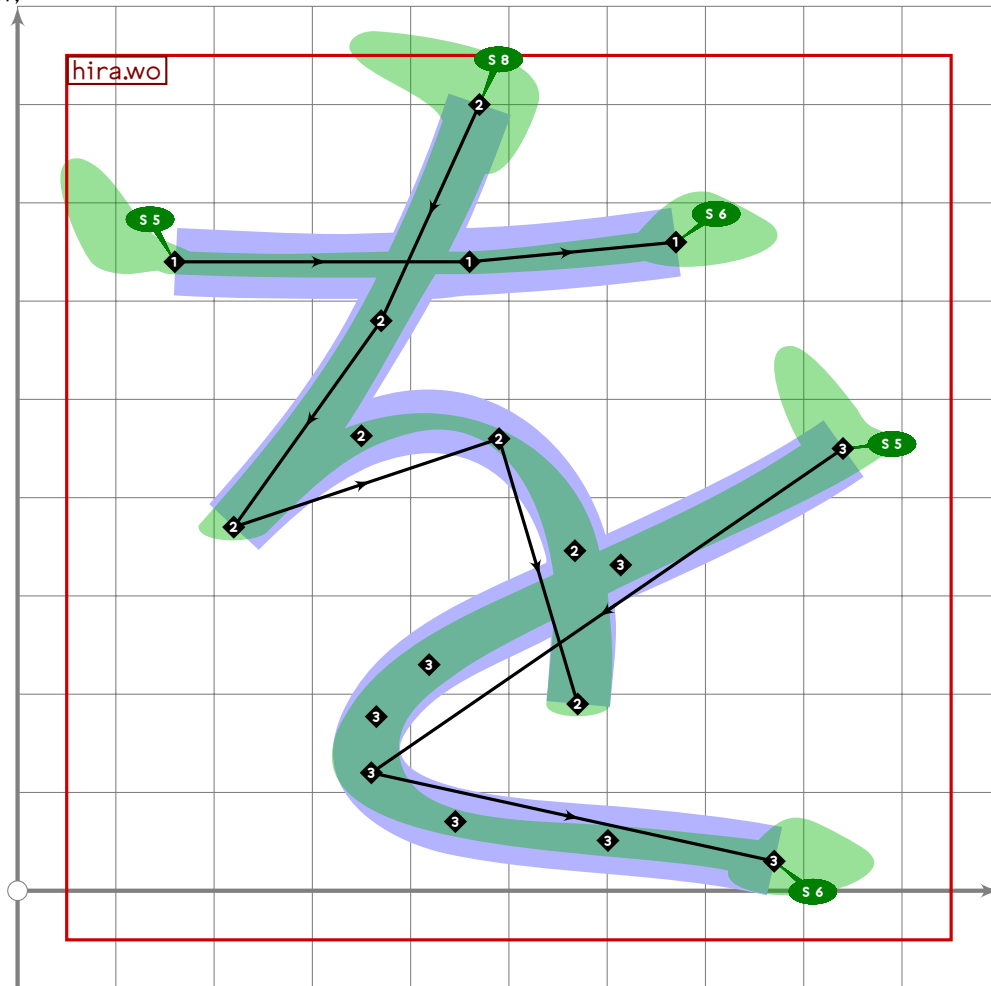
HIRA

U+3092
tsuku.uni3092

```

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

```



```

934
935 vardef hirawo =
936   push_pbox_toexpand("hirawo");
937
938   push_stroke((160,640)..(460,640)..(670,660),
939     (1.3,1.3)–(1.4,1.4)–(1.6,1.6));
940   set_boserif(0,0,5);
941   set_boserif(0,2,6);
942
943   push_stroke((470,800)..(370,580)..{curl 1}{(220,370){curl 0.1}..
944     (490,460)..{curl 0}{(570,190),
945     (1.4,1.4)–(1.3,1.3)–(1.5,1.5)–(1.01,1.01)–(1.4,1.4));
946   set_botip(0,2,0);
947   set_boserif(0,0,8);

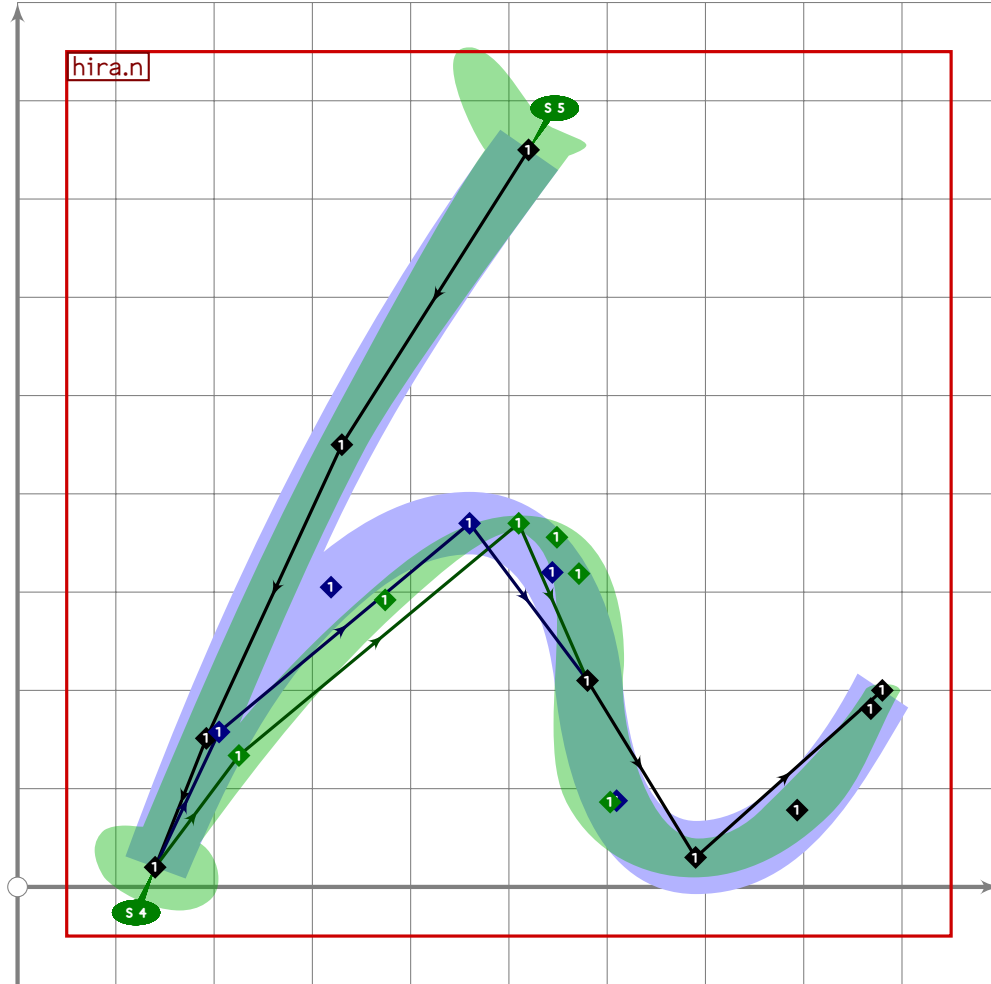
```

HIRA

```

948
949 push_stroke((840,450){curl 0.017}..tension 1 and 2..(360,120)..
950   tension 2 and 1..{curl 0.03}(770,30),
951   (1,1,7)-(14,14)-(1,1,7));
952 set_boserif(0,0,5);
953 set_boserif(0,2,6);
954 expand_pbox;
955 enddef;

```



```

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

```

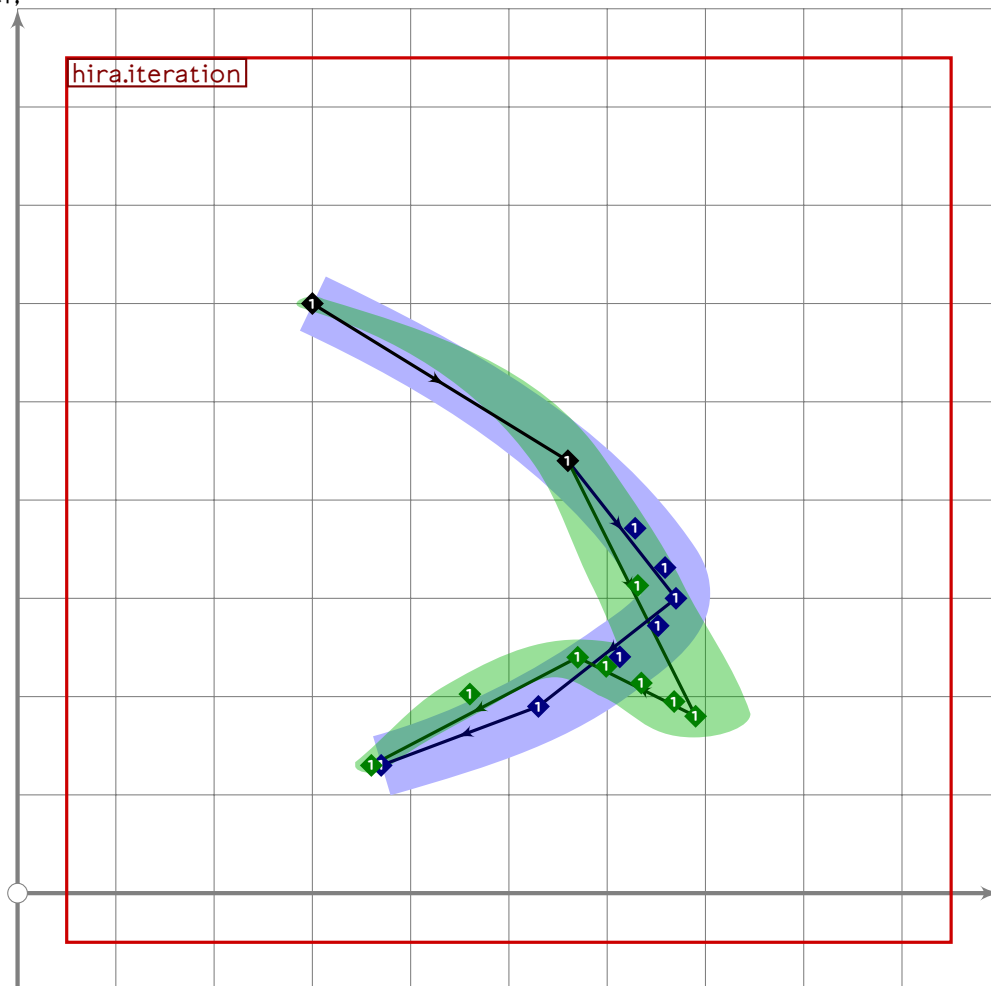
HIRA

U+309D
tsuku.uni309D

```

969 set_boserif(0,0,5);
970 set_boserif(0,3,4);
971 expand_pbox;
972 enddef;

```



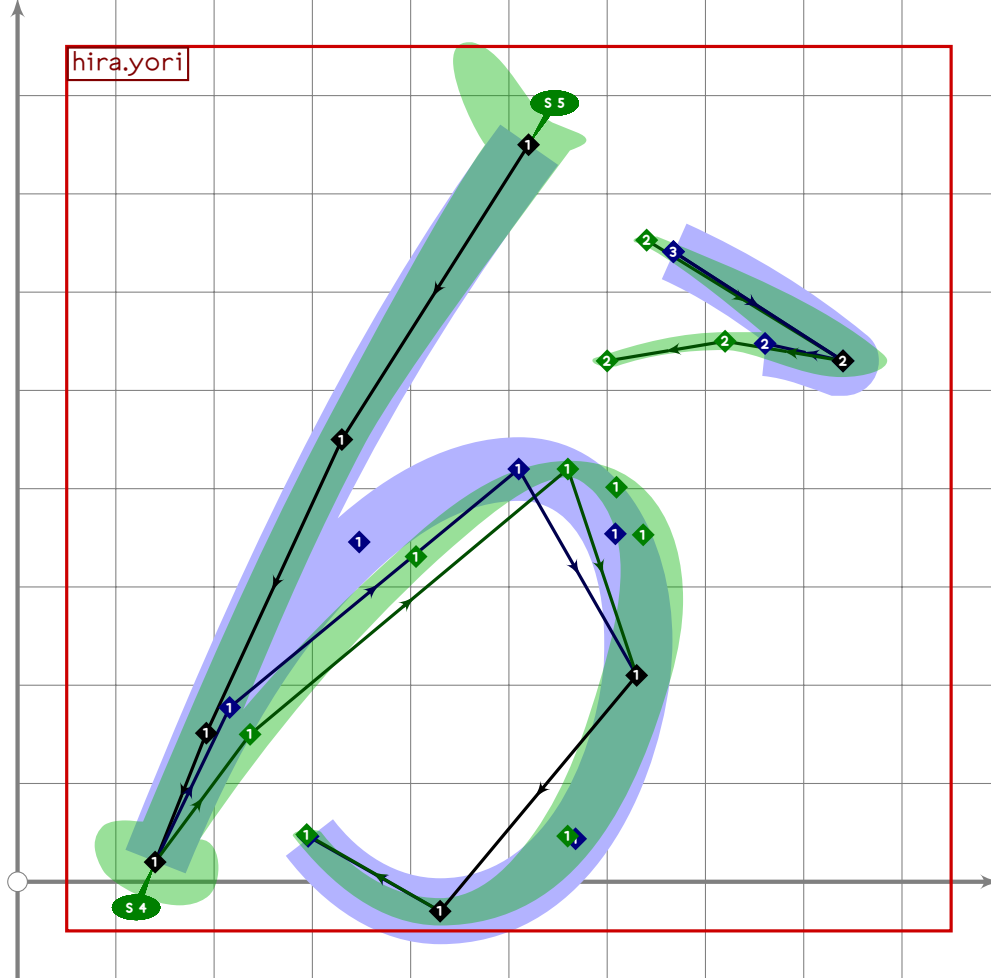
```

973
974 vardef hira.iteration =
975   push_pbox_toexpand("hira.iteration");
976
977   push_stroke(begingroup
978     save ripx,ripy;
979     path ripx,ripy;
980     ripx:=(300,600){curl 0.2}..(560,440)..
981       tension 1.5 and 2..(670,300)..
982       tension 2 and 1.5..(530,190){curl 0.2}(370,130);
983     ripy:=(300,600){curl 0.2}..(560,440)..
984       tension 1.5.{curl 1}(690,180){curl 1}..
985       tension 2 and 1.5..(570,240)..tension 14.{curl 0}(360,130);
986     interpath(mincho,ripx,ripy)
987   endgroup,
988   (1,1)–(1.5,1.5)–(2,2)–(1.9,1.9)–(1,1));
989   set_botip(0,2,0);

```

HIRA

```
990 expand_pbox;
991 enddef;
```



```
992
993 vardef hira.yori =
994   push_pbox_toexpand("hira.yori");
995
996   push_stroke((520,750)..(330,450)..{curl 0.2}(140,20){curl 0.1}..
997     tension (1.2+0.6*mincho)..(510+50*mincho,420){right}..
998     (630,210)..(430,-30){left}..tension 1.3..(290,600)..
999     {curl 0.1}(840,530){curl 1}..(720,550)..(600,530),
1000   (1.7,1.7)-(1.2,1.2)-(1.3,1.3)-
1001   (1.1,1.1)-
1002   (1.5,1.5)-(1.5,1.5)-(-1,-0.4)-
1003   (1.8,1.6)-(1.6,0.7)-(1.9,0));
1004   replace_strokep(0)(insert_nodes(oldp)(1.7,2.3));
1005   replace_strokeq(0)(insert_nodes(oldq)(1.7,2.3));
1006   set_botip(0,3,0);
1007   set_botip(0,9,0);
1008   set_boserif(0,0,5);
1009   set_boserif(0,3,4);
1010   expand_pbox;
```

HIRA

1011 enddef;

HIRA

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

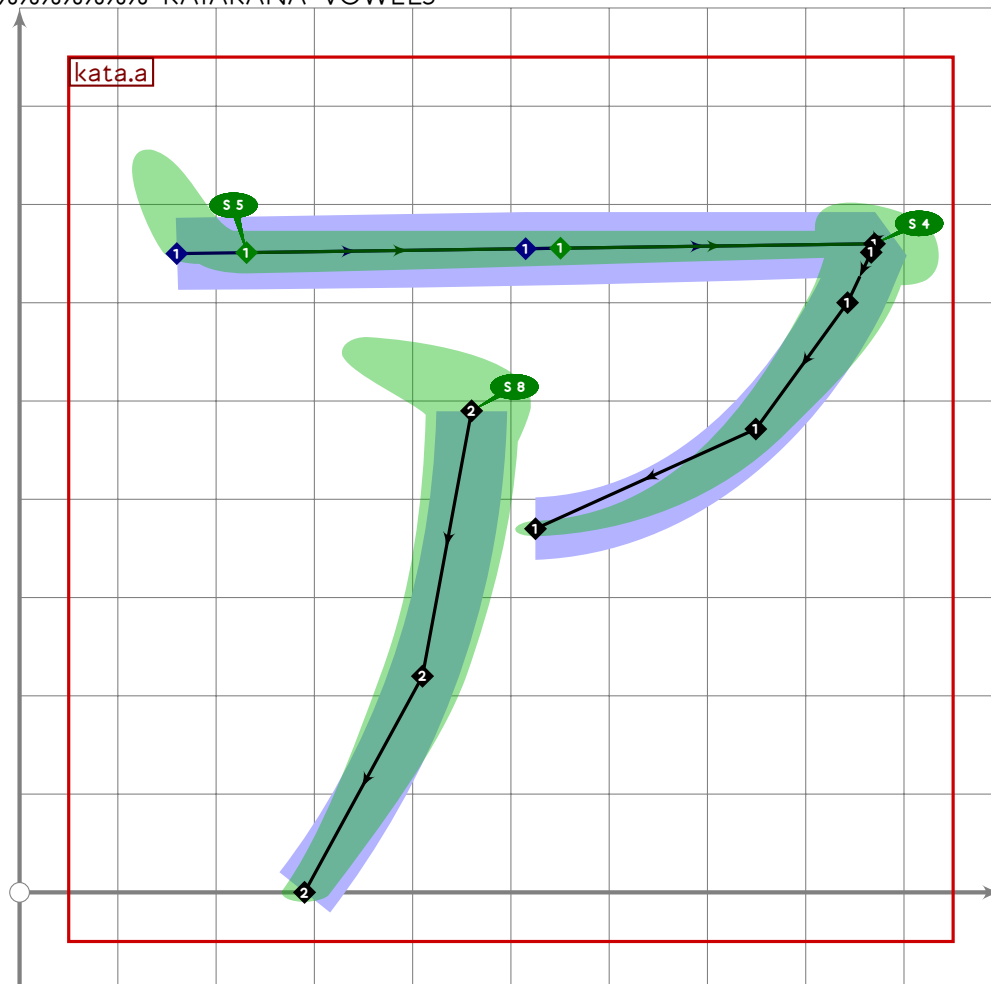
```

1 %
2 % Katakana for Tsukurimashou
3 % Copyright (C) 2011, 2012, 2013, 2017, 2019 Matthew Skala
4 %
5-29 [Standard copyright notice]
30
31 inclusion_lock(katakana);
32
33
34

```

Katakana Vowels

35 %%%%%%%%%%%%%%%%% 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

```

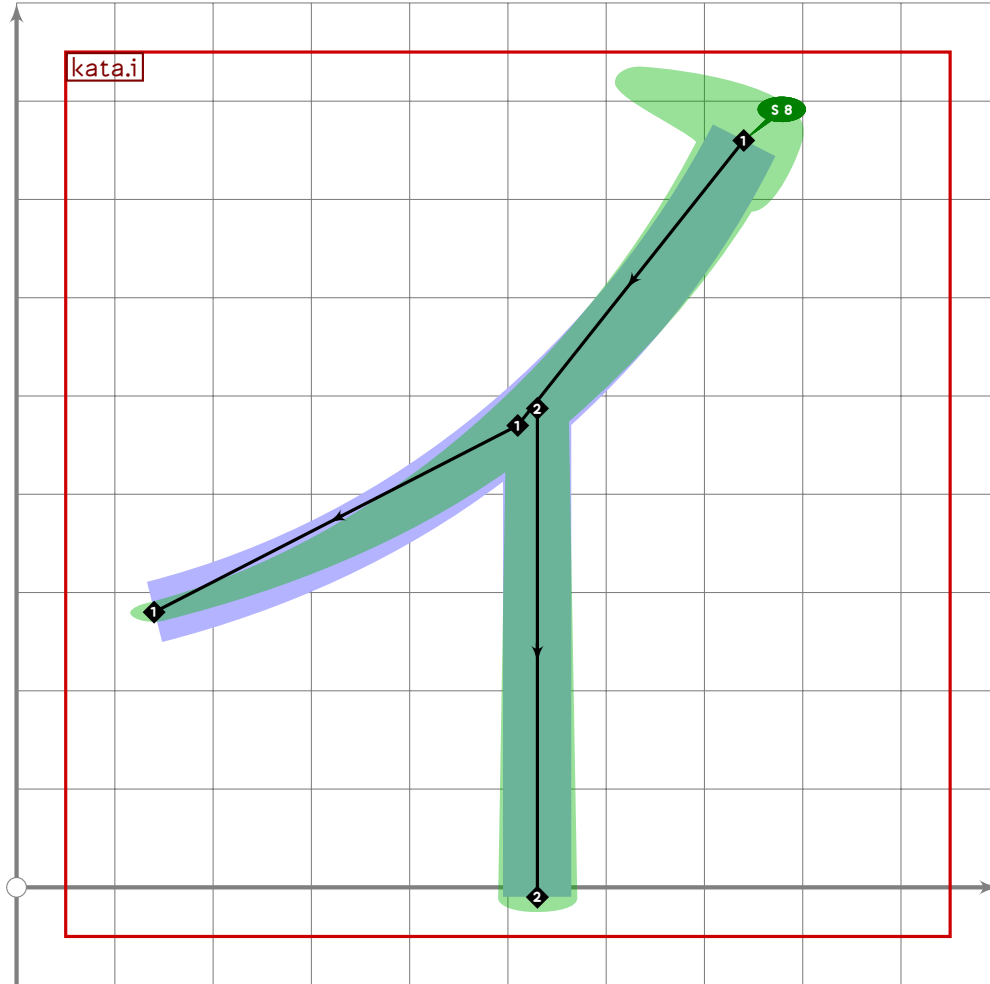
KATA

U+30A4
tsuku.uni30A4

```

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;

```

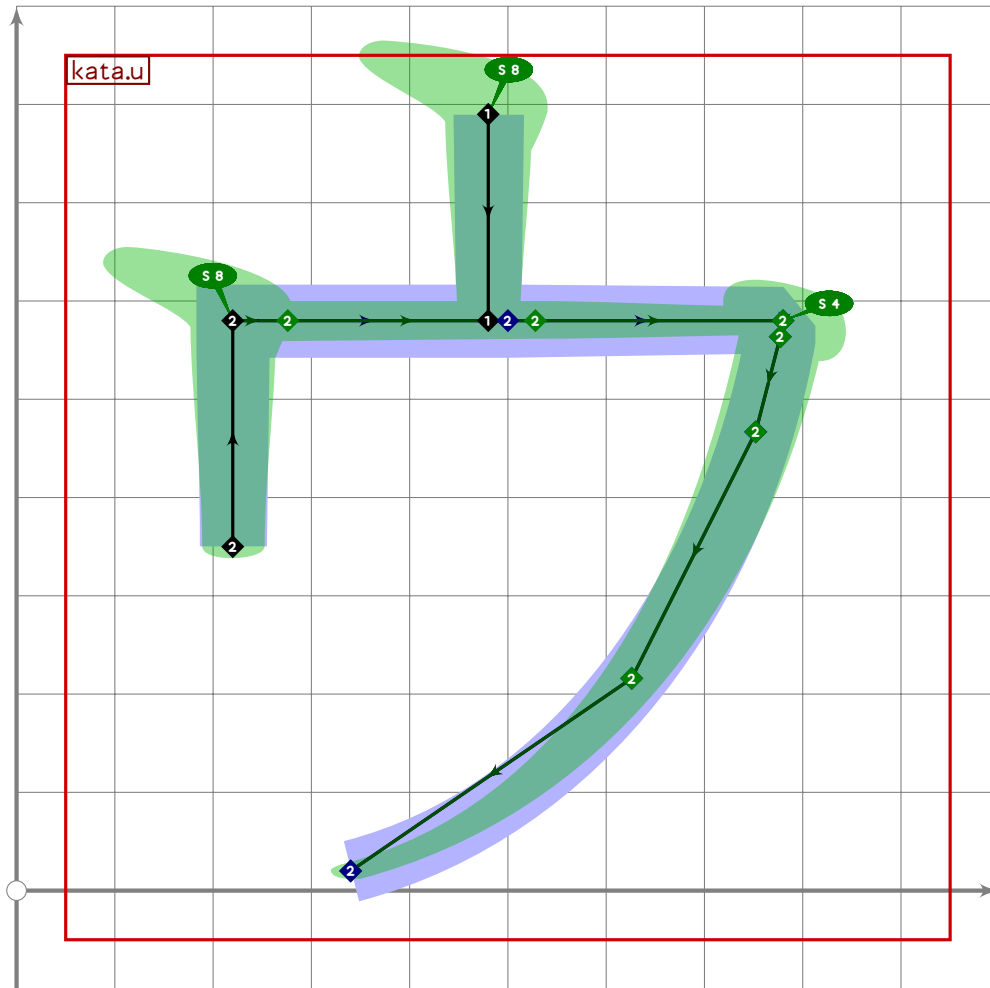


```

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_stroke(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);

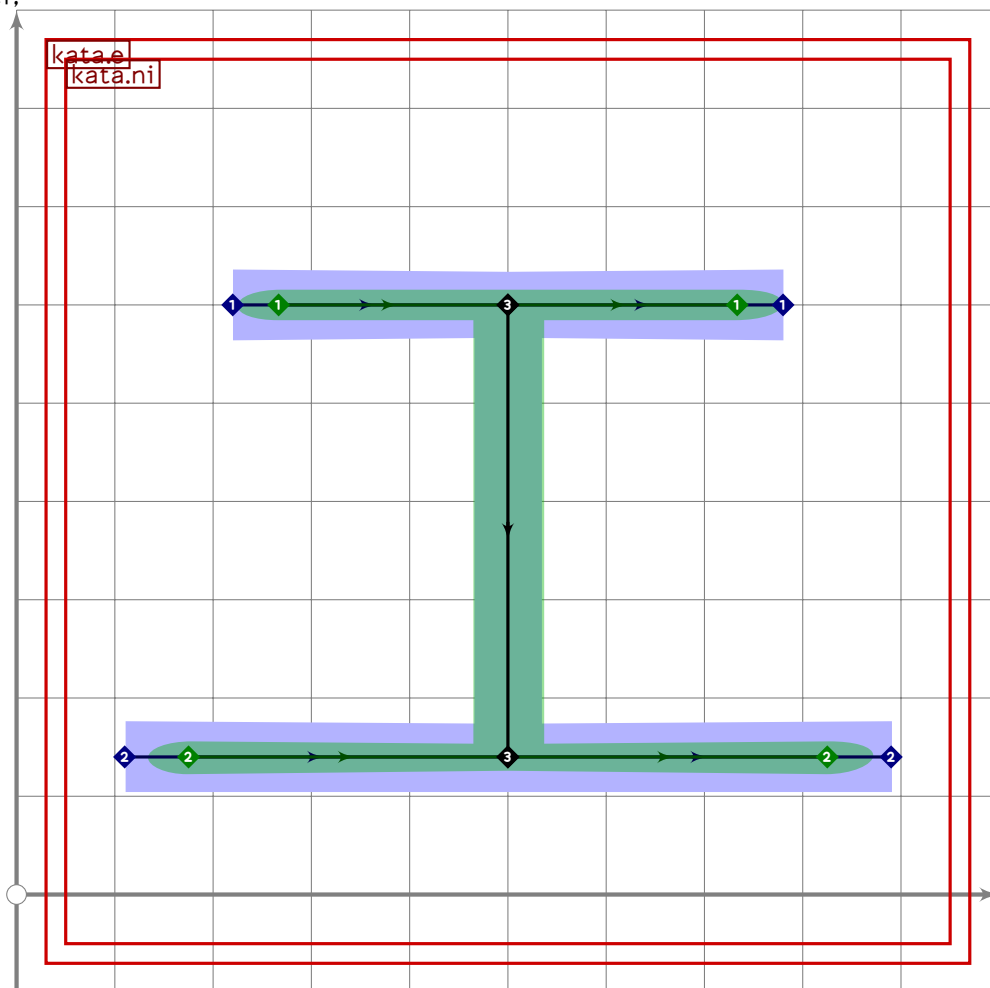
```

U+30A8
tsuku.uni30A8

```

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;

```

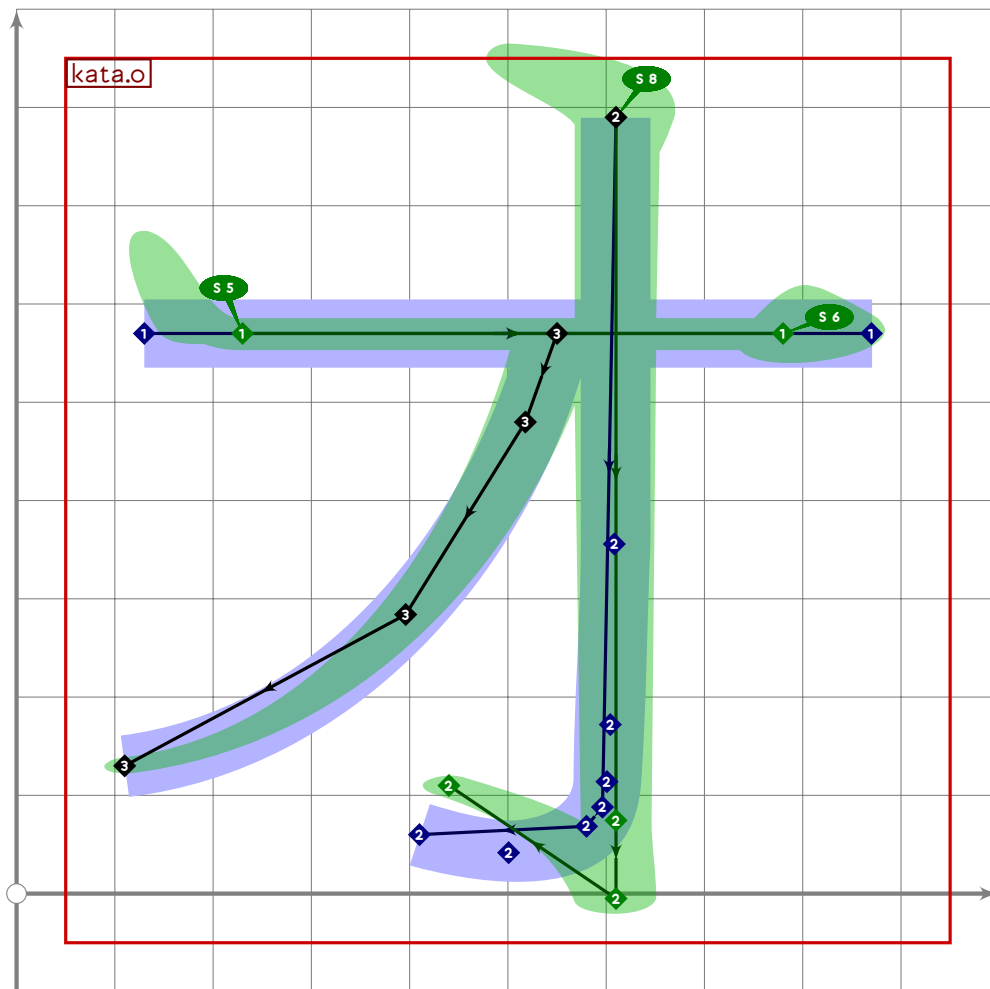


```

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;

```

KATA



```

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

```

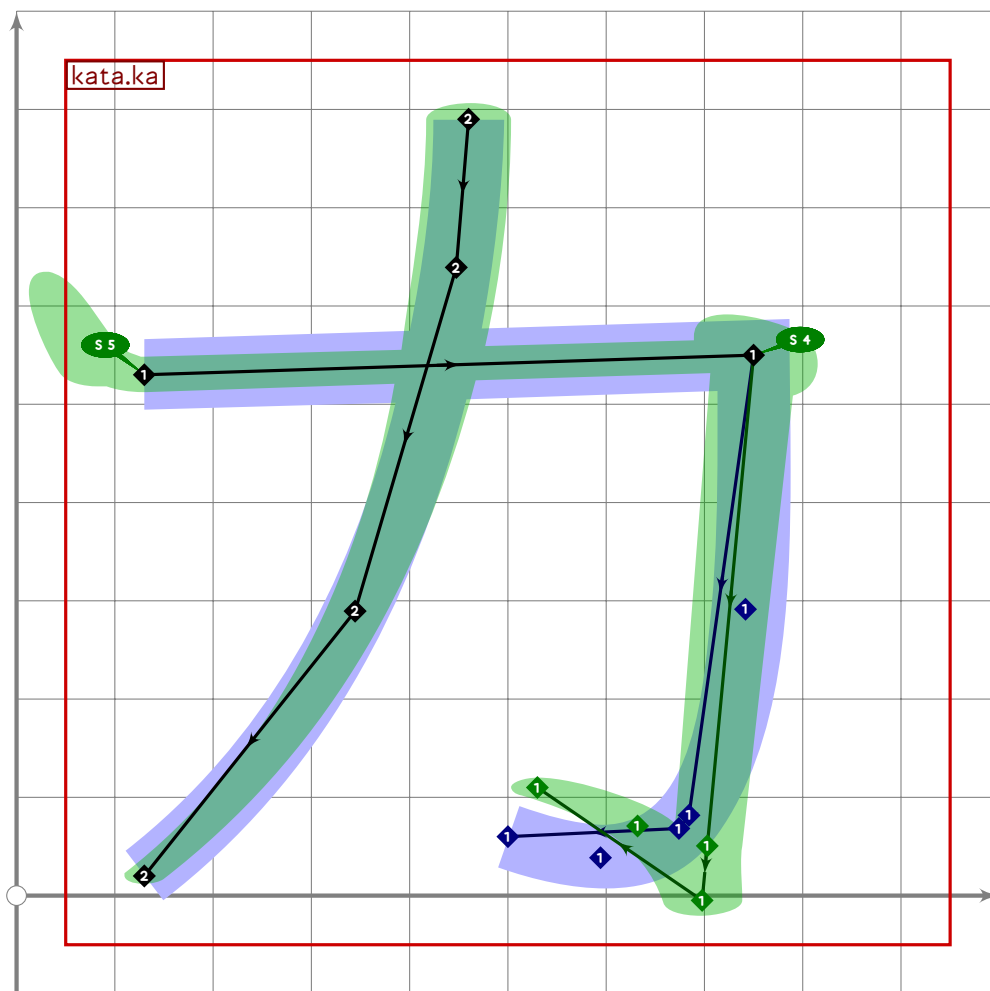
KATA

Katakana Kakikukeko/Gagigugego

```

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

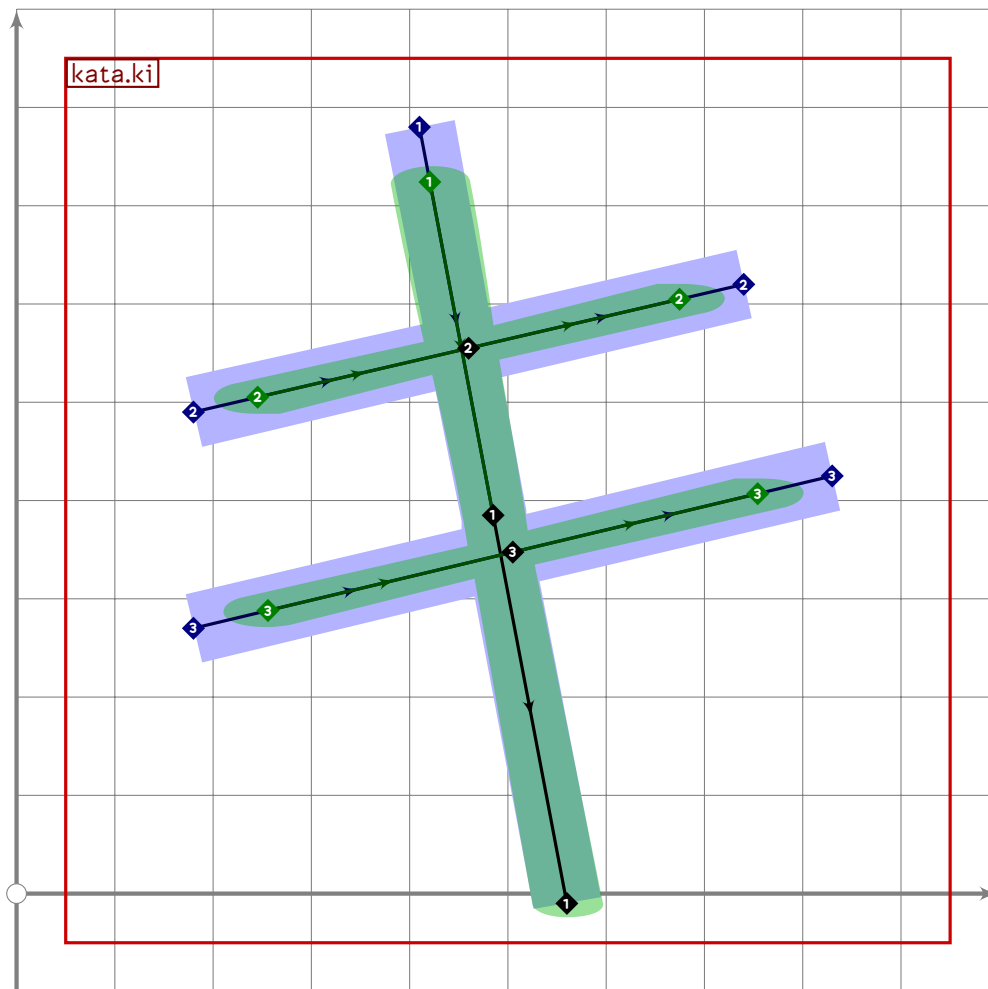
```



```

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;

```



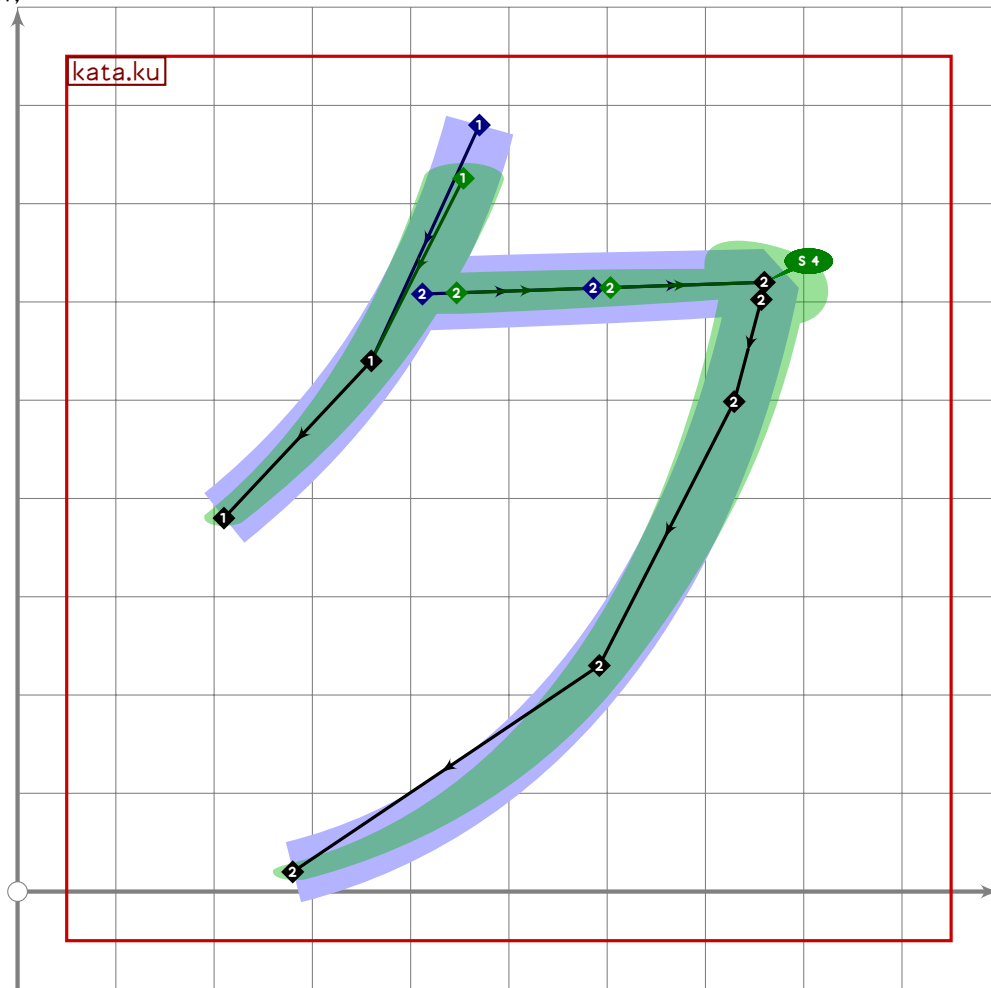
```

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;

```

U+30AF
tsuku.uni30AF

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;

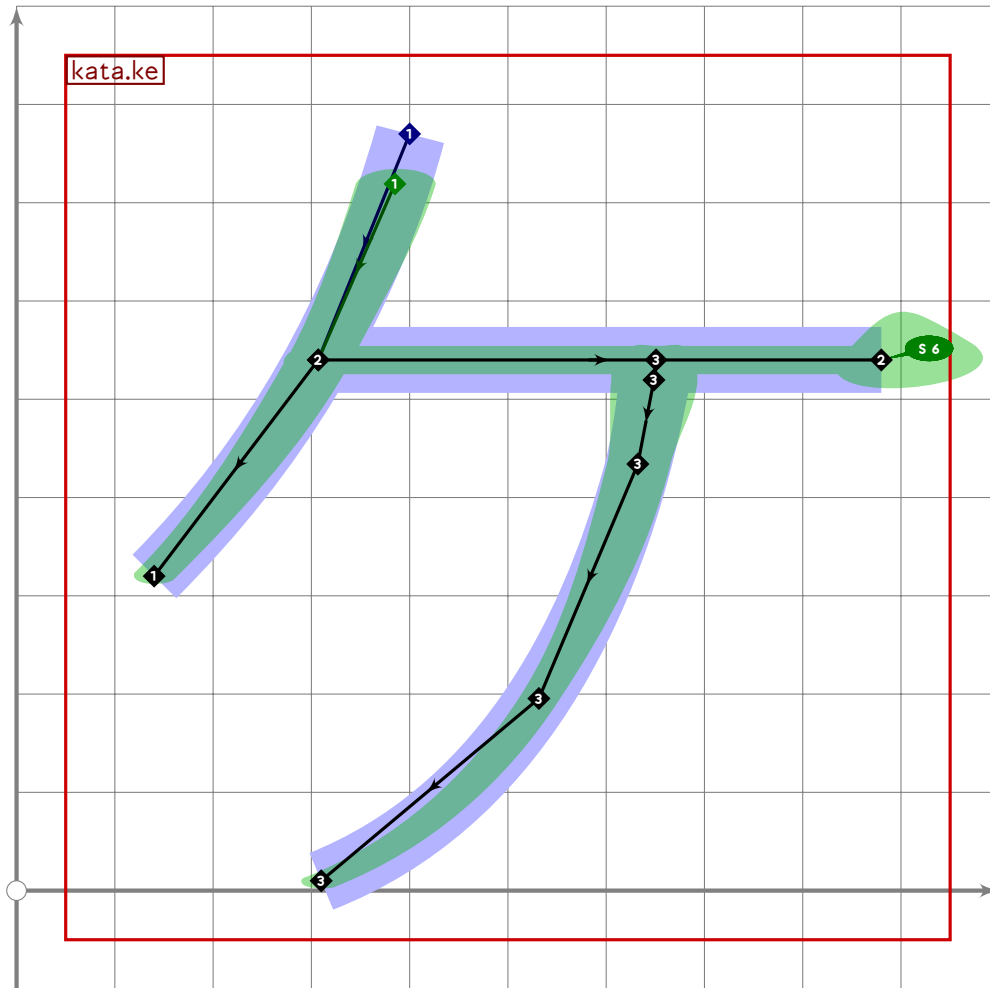
161 kata.fu_stroke(z1,(760,620),(280,20));

162 set_boserif(0,0,whatever);

163 expand_pbox;

164 endif;

KATA

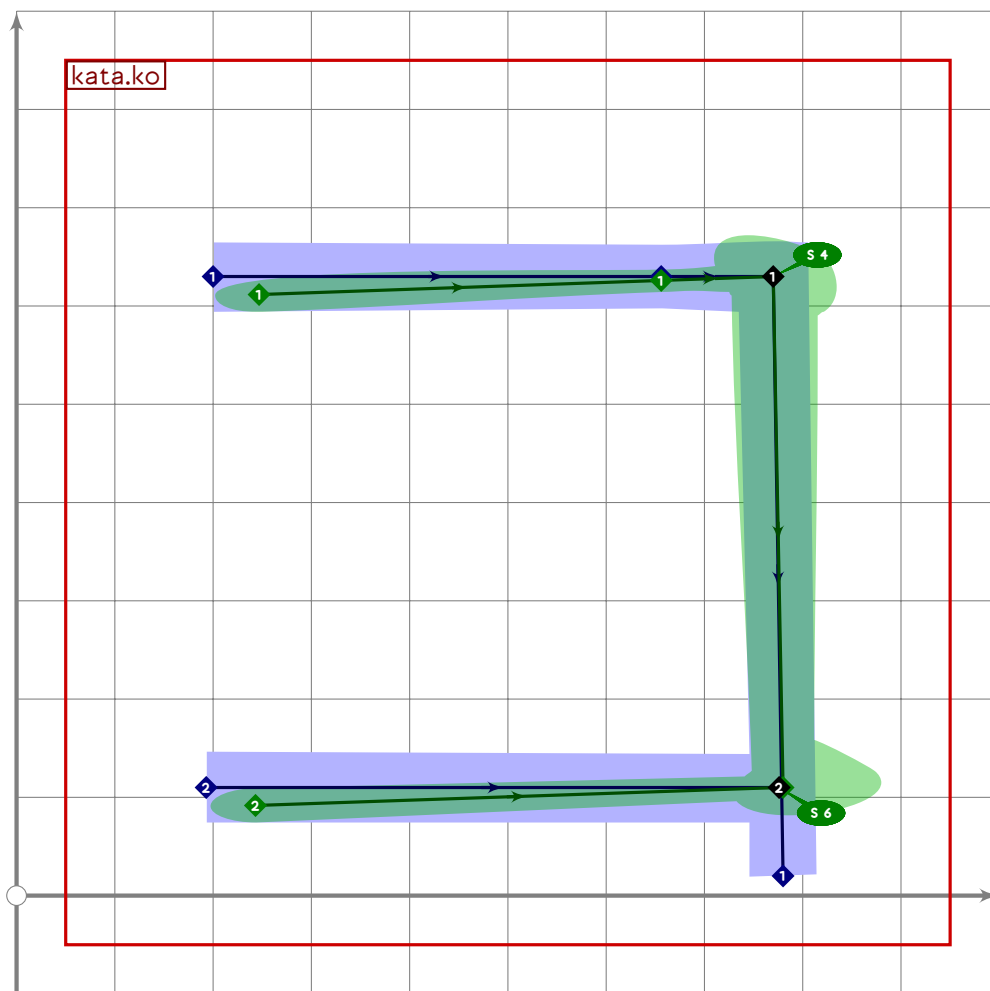


```

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)-(14,14)-(11,1.1));
171   set_boserif(0,0,5);
172
173   z1=(get_stroke(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_stroke(0)(insert_nodes(oldp)(0.2));
179   expand_pbox;
180 enddef;

```

KATA



```

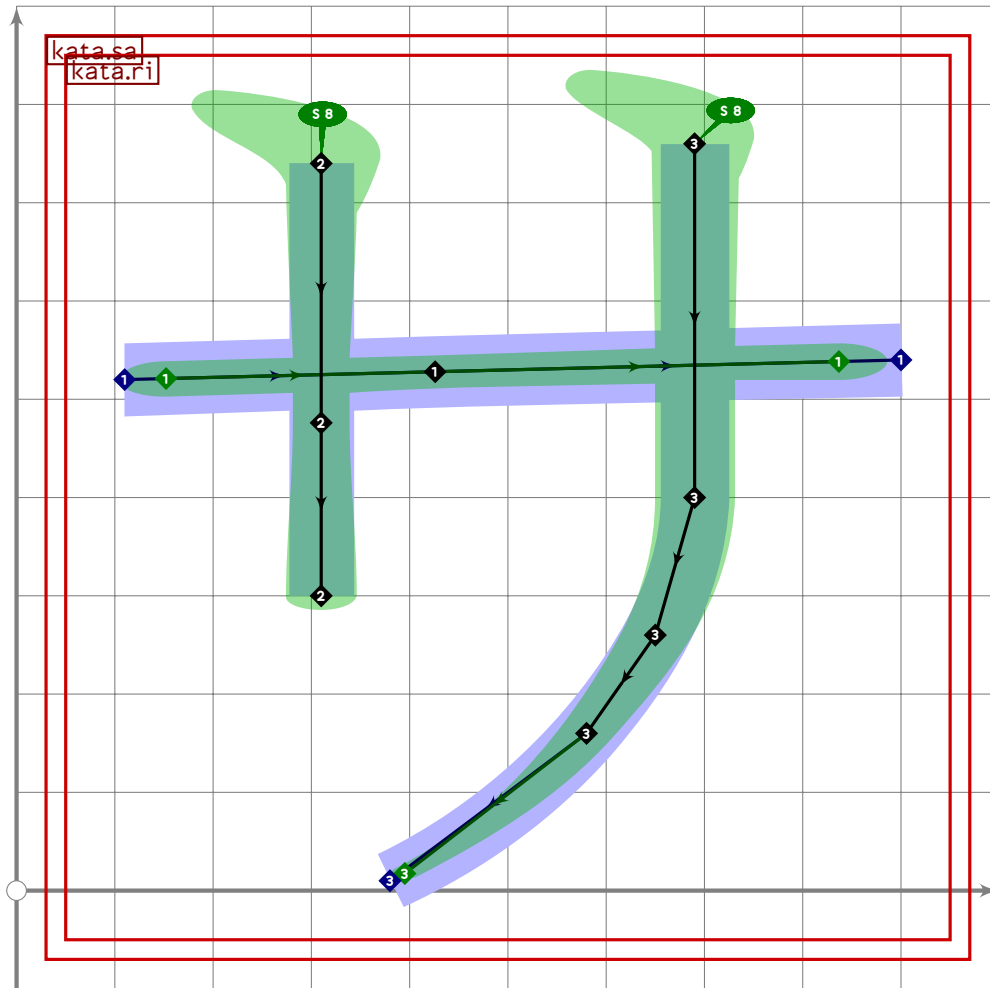
181
182 vardef kata.ko =
183   push_pbox__toexpand("kata.ko");
184
185   push_stroke((200,630-20*mincho)-(770,630)-(780,mincho[20,110]),
186     (0.78,2.83)-(1.3,1.3)-(1.7,1.7)-(1.4,1.4));
187   replace_strokep(0)(insert_nodes(oldp)(0.8));
188   set_botip(0,2,1);
189   set_boserif(0,0,5);
190   set_boserif(0,2,4);
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

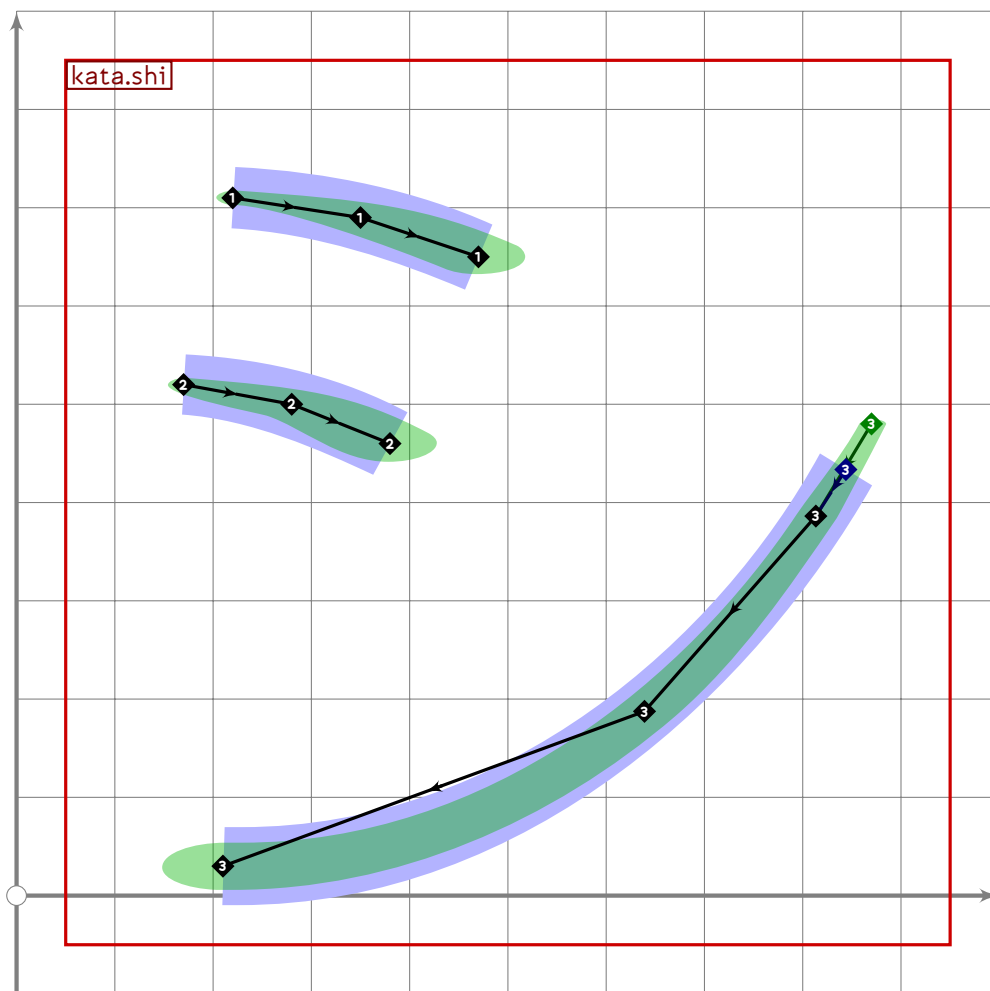
199 %%%%%%%%% KATAKANA SASHISUSES0/ZAJIZUZEZO



```

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;

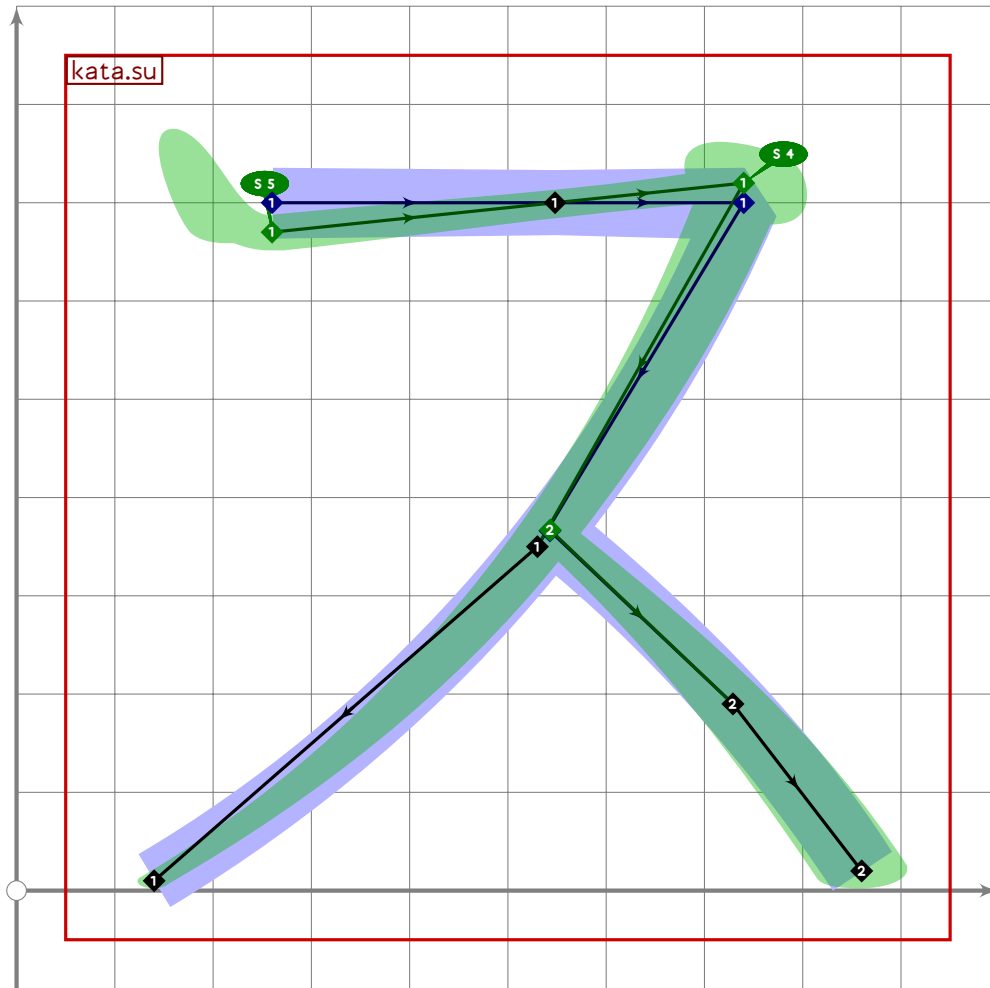
```



```

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)((0,0,0)-(1,1,1)-(1,4,1)-(2,2,2,2));
226   expand_pbox;
227 enddef;

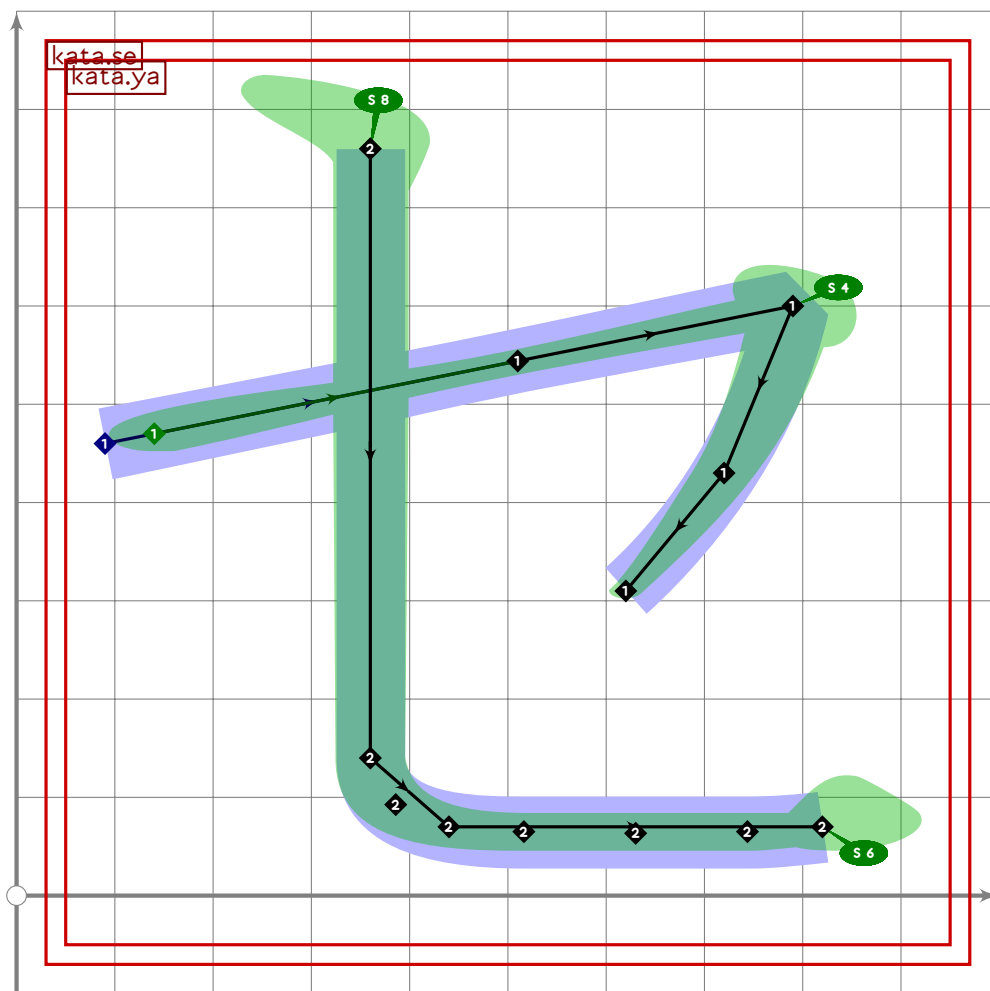
```



```

228
229 vardef kata.su =
230   push_pbox__toexpand("kata.su");
231
232   push_stroke((260,700-30*mincho)-(740,700+20*mincho)..(530,350)..(140,10),
233     (1.8,1.8)-(1.3,1.3)-(1.7,1.7)-(1.4,1.4)-(1,1));
234   replace_stroke(0)(insert_nodes(oldp)(0.6));
235   set_botip(0,2,0);
236   set_boserif(0,0,5);
237   set_boserif(0,2,4);
238
239   push_stroke((point 2.95 of get_stroke(0))..(729,190)..(860,20),
240     (1.2,1.2)-(1.6,1.6)-(1.8,1.8));
241   expand_pbox;
242 enddef;

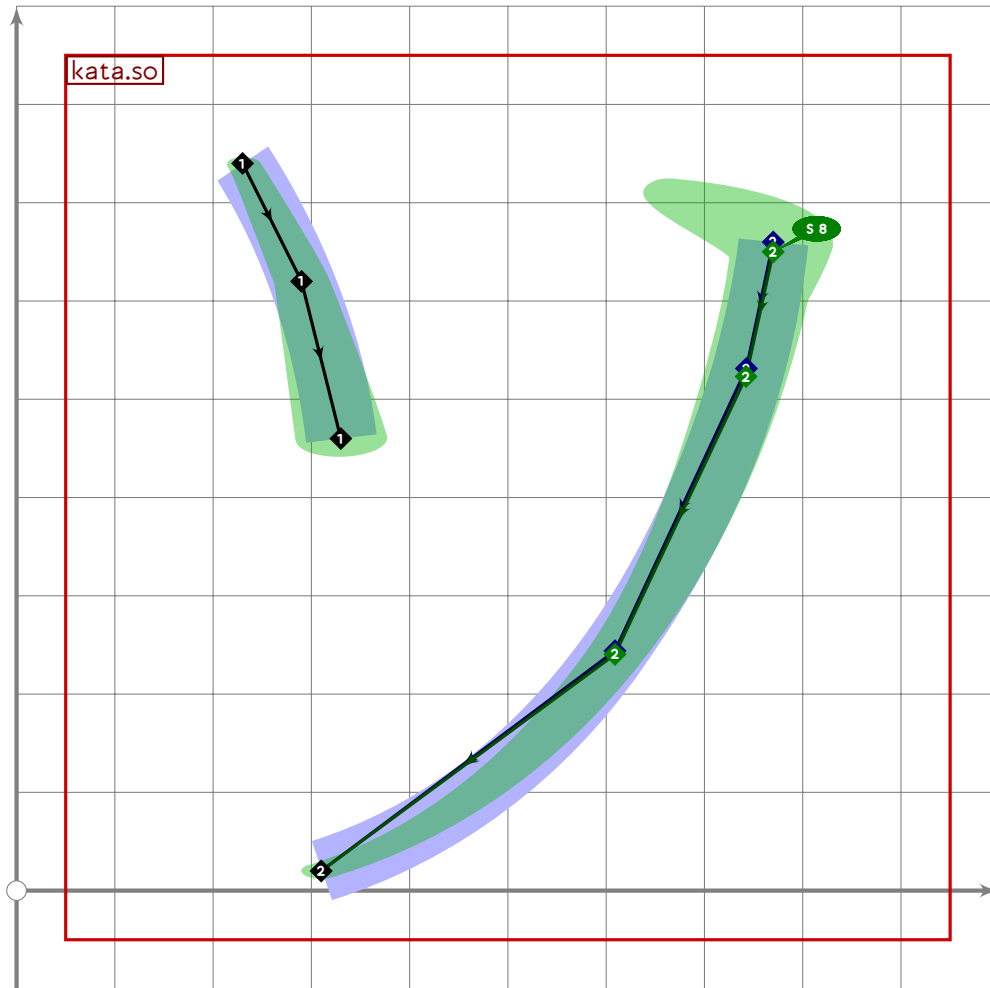
```



```

243
244 vardef kata.se =
245   push_pbox_toexpand("kata.se");
246
247   kata.ya;
248
249   replace_strokep(-1)(oldp shifted (-30,0));
250
251   replace_strokep(0)((360,760)-(360,140){dir 274}..
252     (440,70)..tension 21..(820,70));
253   replace_strokeq(0)((1.6,1.6)-(1.5,1.5)-(1.9,1.9)-(1.8,1.8));
254   set_boserif(0,0,8);
255   set_boserif(0,3,6);
256   expand_pbox;
257 enddef;

```



```

258
259 vardef kata.so =
260   push_pbox__toexpand("kata.so");
261
262   push_stroke((230,740)..(290,620)..(330,460),
263     (1,1)..(1.3,1.3)..(1.8,1.8));
264
265   kata.no_stroke((770,660-10*mincho),(310,20));
266   set_boserif(0,0,8);
267   expand_pbox;
268 enddef;
269

```

KATA

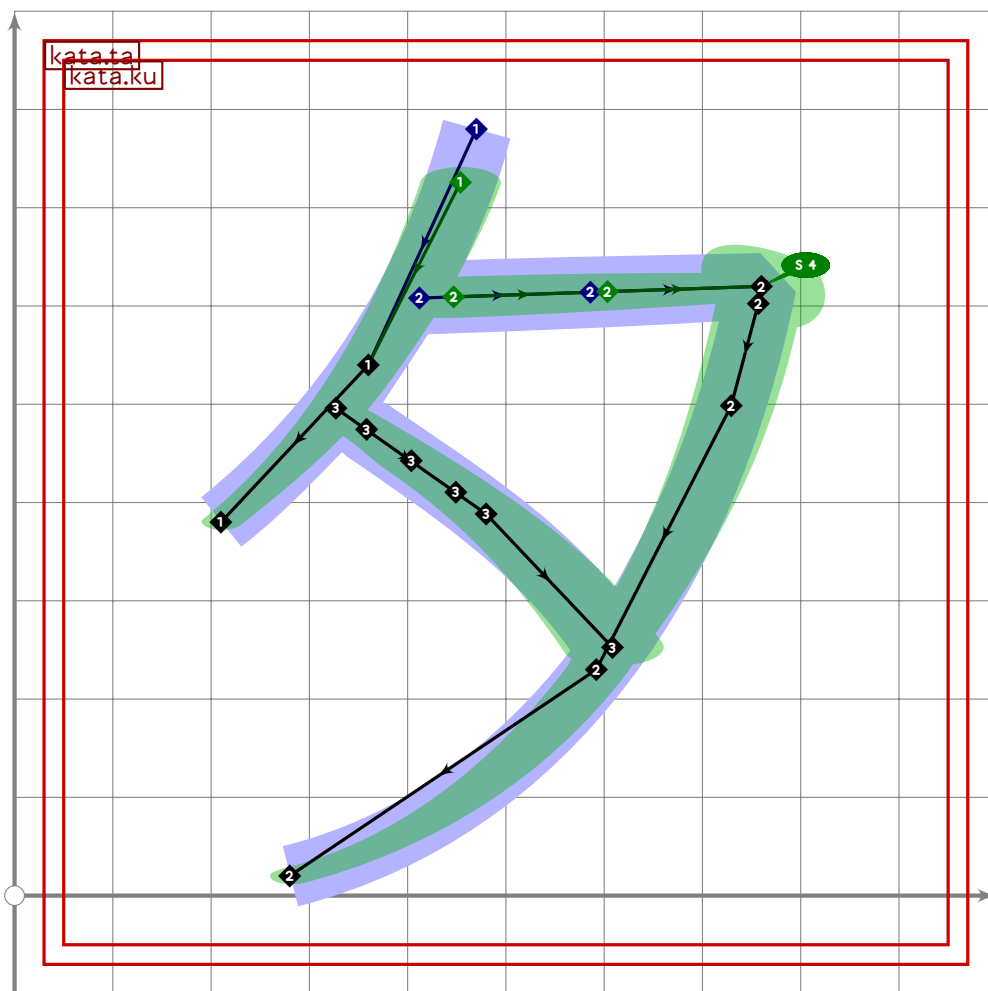
Katakana Tachitsuteto/Dajizudedo

```

270 %%%%%%%%% KATAKANA TACHITSUTETO/DAJIZUDED0

```

U+30BF
tsuku.uni30BF

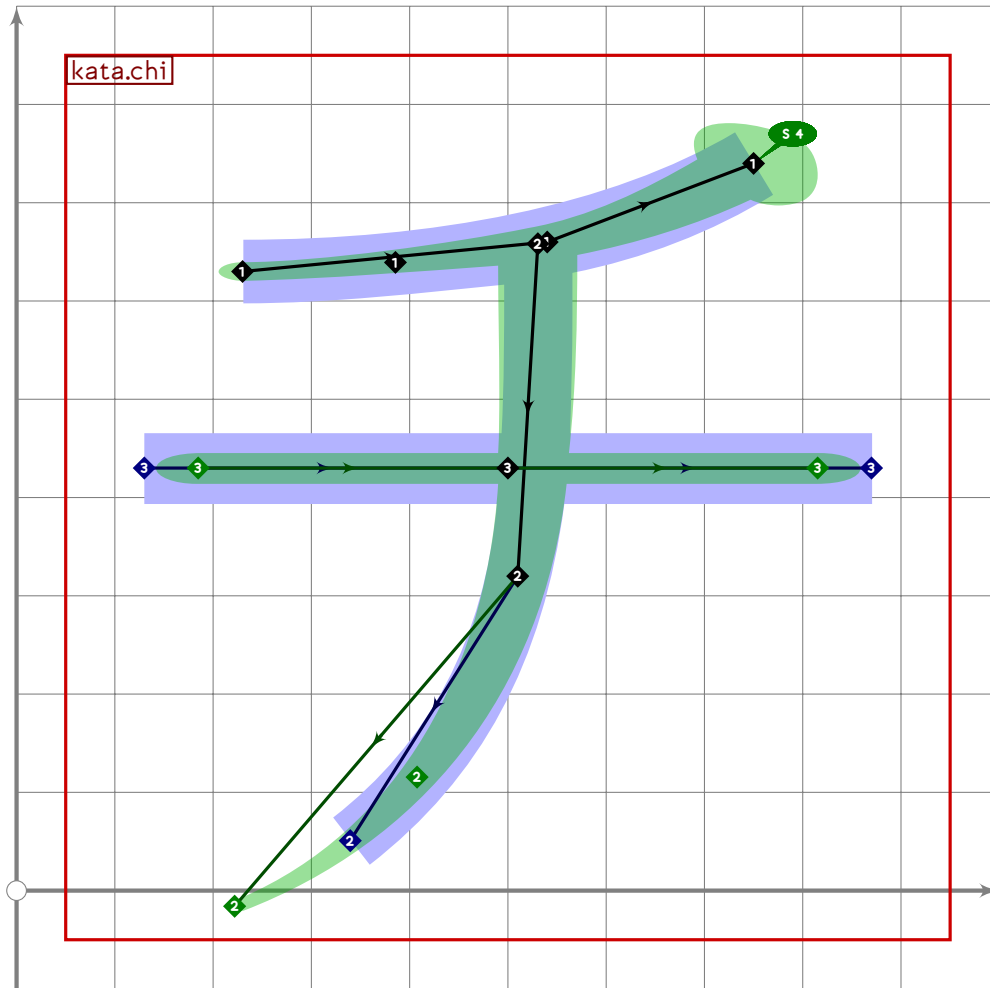


```

271
272 vardef kata.ta =
273   push_pbox__toexpand("kata.ta");
274
275   kata.ku;
276
277   numeric x[],y[];
278   z1=point 1.25 of get_stroke(-1);
279   z3=point 4.9 of get_stroke(0);
280   z2=(0.5[z1,z3])+0.05*((z3-z1) rotated 90);
281   push_stroke(z1..tension 2..z2..z3,
282     (1.2,1.2)-(1.6,1.6)-(19,1.9));
283   expand_pbox;
284 enddef;

```

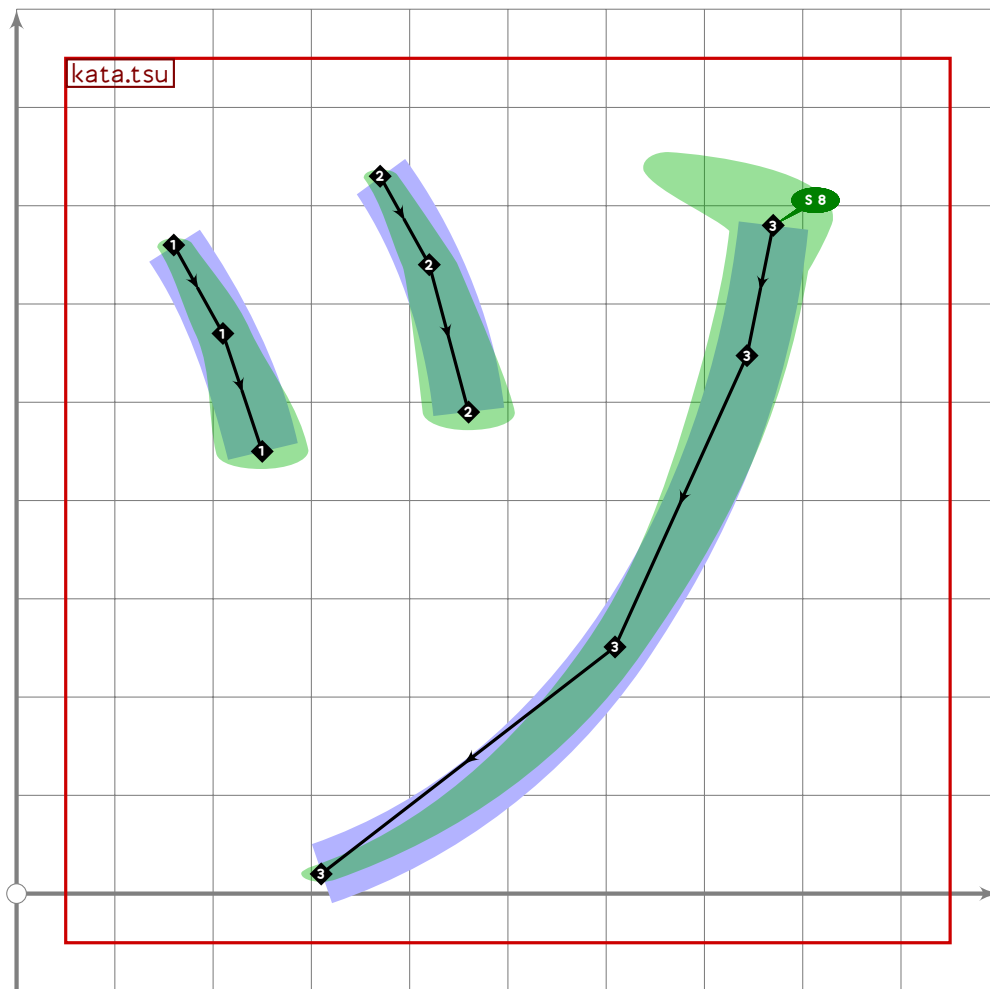
KATA



```

285
286 vardef kata.chi =
287   push_pbox__toexpand("kata.chi");
288
289   push_stroke((230,630)..tension 1.3..(540,660)..(750,740),
290     (1,2,1,2)-(1,7,1,7)-(2,2));
291   set_boserif(0,2,4);
292
293   kata.na_centre;
294   replace_stroke(0)(subpath (xpart (oldp intersectiontimes
295     get_stroke(-1)),infinity) of oldp);
296
297   push_stroke((130,430)-(870,430),
298     (0,7,2,7)-(1,6,1,6)-(0,7,2,7));
299   replace_stroke(0)(insert_nodes(oldp)(0.5));
300   set_boserif(0,0,5);
301   set_boserif(0,2,6);
302   expand_pbox;
303 enddef;

```



```

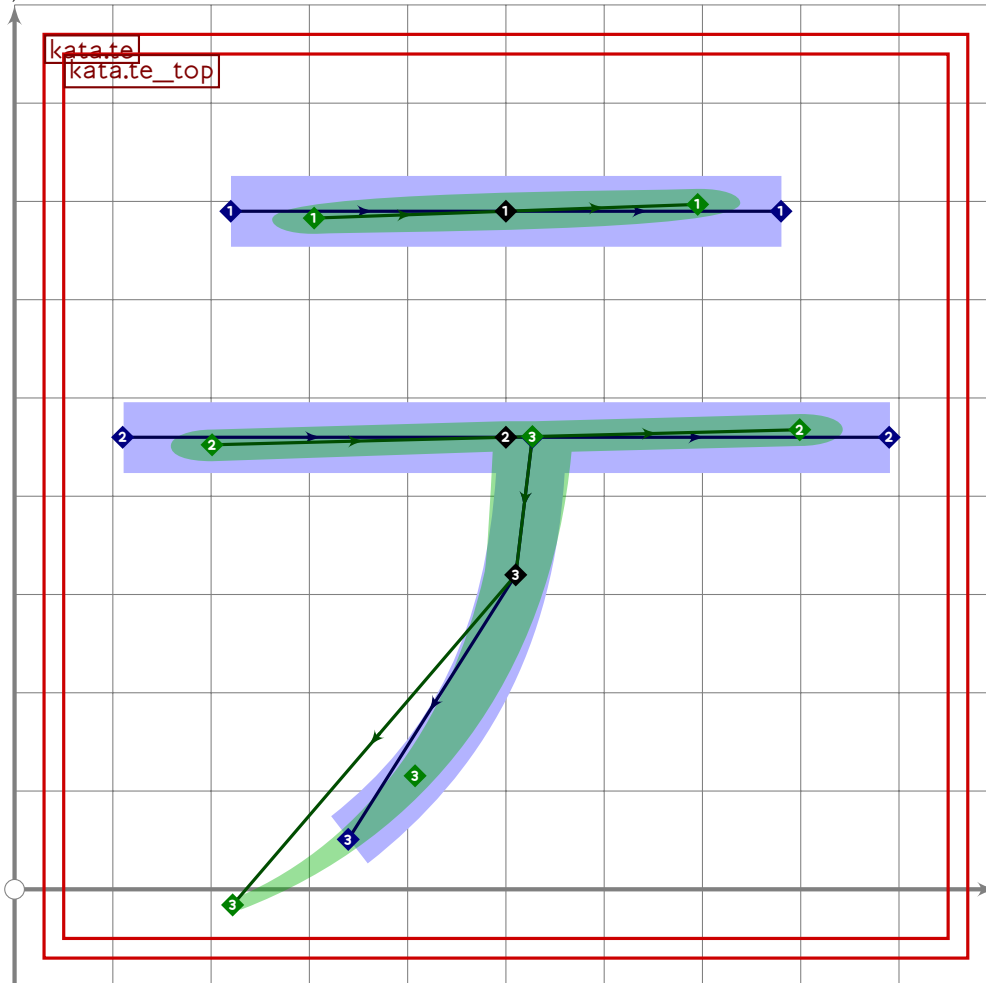
304
305 vardef kata.tsu =
306   push_pbox_toexpand("kata.tsu");
307
308   push_stroke((160,660)..(210,570)..(250,450),
309     (1,1)..(1.3,1.3)..(1.8,1.8));
310
311   push_stroke((370,730)..(420,640)..(460,490),
312     (1,1)..(1.3,1.3)..(1.8,1.8));
313
314   kata.no_stroke((770,680),(310,20));
315   set_boserif(0,0,8);
316   expand_pbox;
317 enddef;
318
319 vardef kata.te_top =
320   push_pbox_toexpand("kata.te_top");
321
322   push_stroke((220,690-10*mincho)-(780,690+10*mincho),
323     (0.5,2.9)-(1.6,1.6)-(0.5,2.9));
324   replace_strokep(0)(insert_nodes(oldp)(0.5));

```

```

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

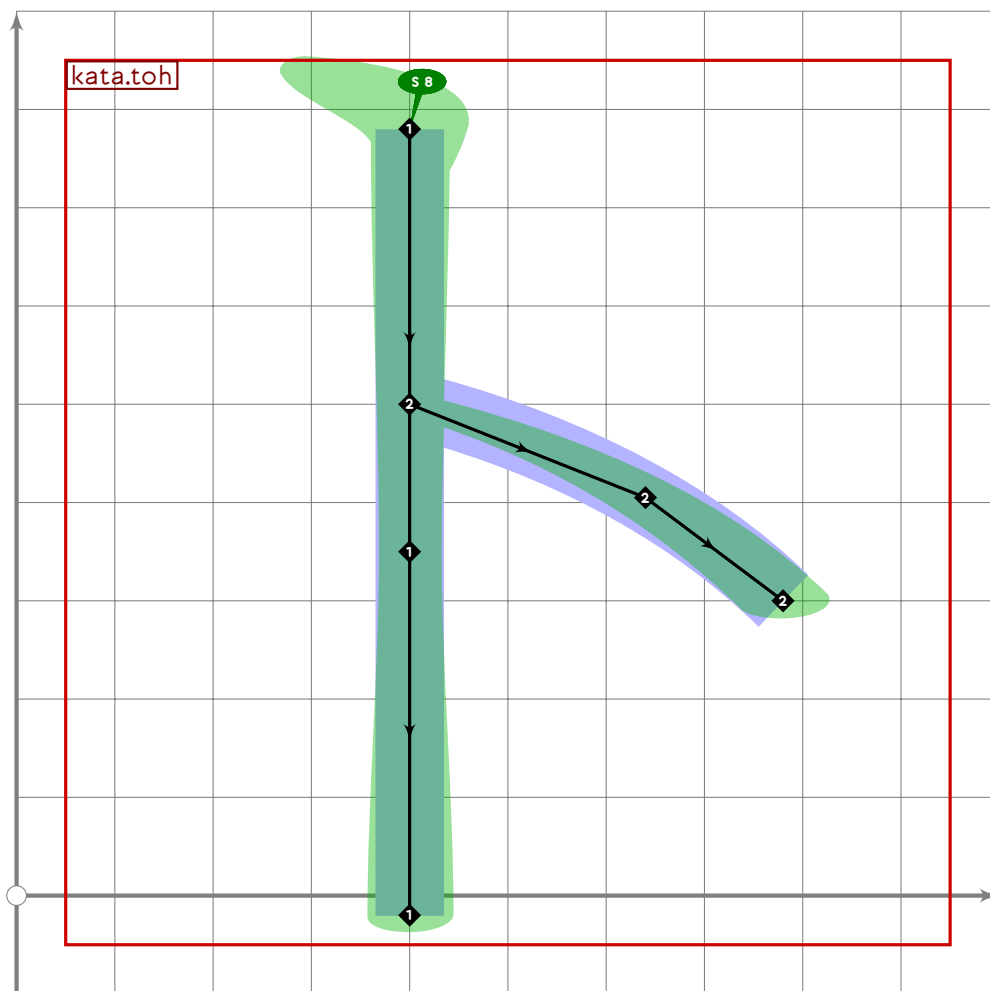
```



```

335
336 vardef kata.te =
337   push_pbox_toexpand("kata.te");
338
339   kata.te_top;
340
341   kata.na_centre;
342   replace_strokep(0)(subpath (xpart (oldp intersectiontimes get_strokep(-1)),
343     infinity) of oldp);
344   expand_pbox;
345 enddef;

```



```

346
347 vardef kata.toh =
348   push_pbox_toexpand("kata.toh");
349
350   push_stroke((400,780)–(400,350)–(400,20),
351     (1.6,1.6)–(1.4,1.4)–(1.7,1.7));
352   set_boserif(0,0,8);
353
354   push_stroke((400,500)..tension 11..(640,405)..(780,300),
355     (1.3,1.3)–(1.6,1.6)–(1.8,1.8));
356   expand_pbox;
357 enddef;
358

```

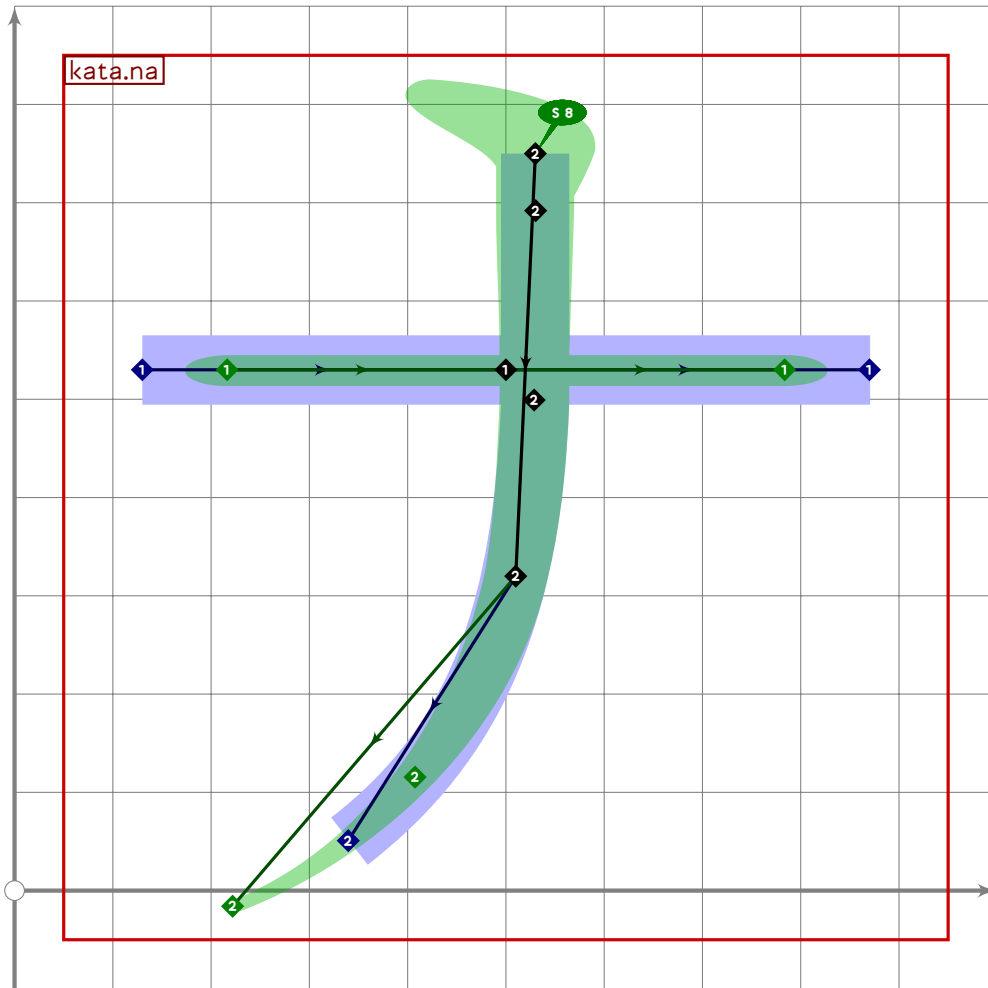
KATA

Katakana Naninuneno

```

359 %%%%%%%%% KATAKANA NANINUNENO
360
361 vardef kata.na_centre =
362   push_stroke((530,750){down}..tension 1.2..(510,320)..(180,30),
363     (1.6,1.6)–(1.4,1.4)–(0.78,0.78));
364 enddef;

```

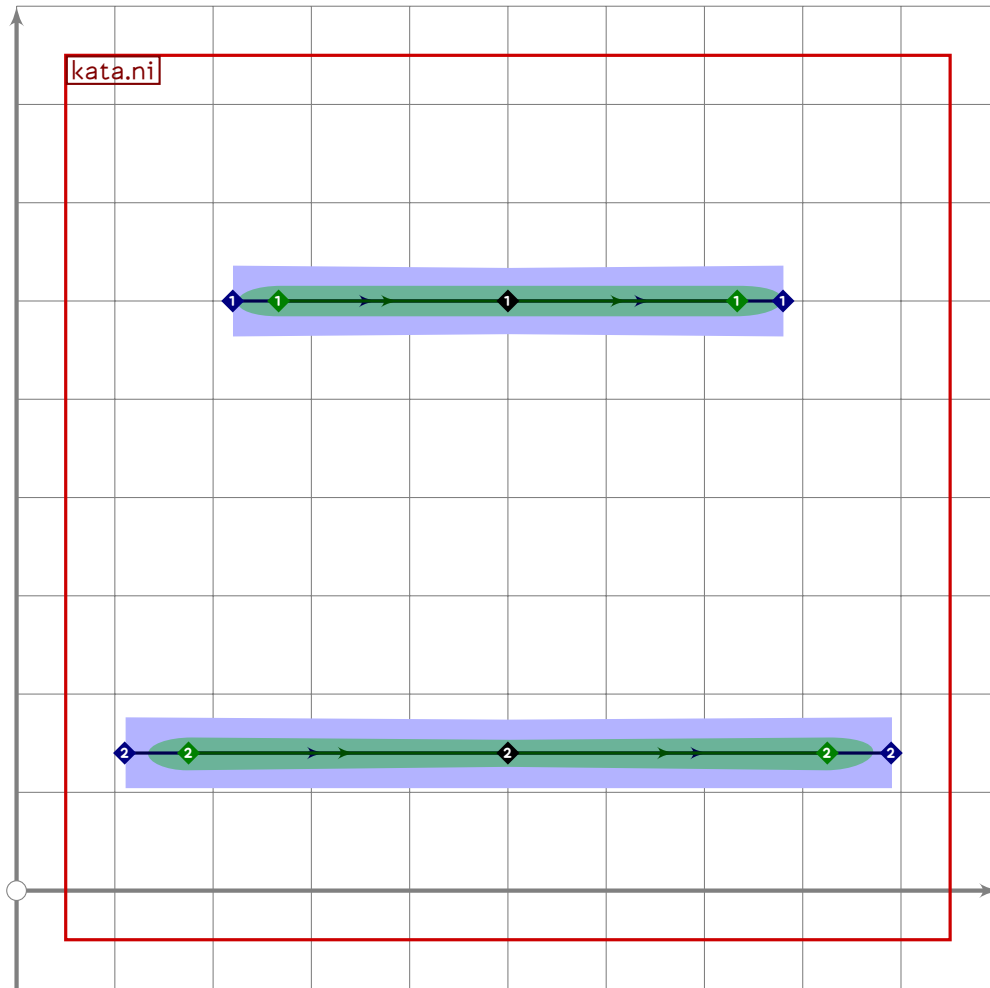


```

365
366 vardef kata.na =
367   push_pbox_toexpand("kata.na");
368
369   push_stroke((130,530)–(870,530),
370     (0,6,2,8)–(1,6,1,6)–(0,6,2,8));
371   replace_strokep(0)(insert_nodes(oldp)(0.5));
372   set_boserif(0,0,5);
373   set_boserif(0,2,6);
374
375   kata.na_centre;
376   set_boserif(0,0,8);
377   expand_pbox;
378 enddef;

```

U+30CB
tsuku.uni30CB

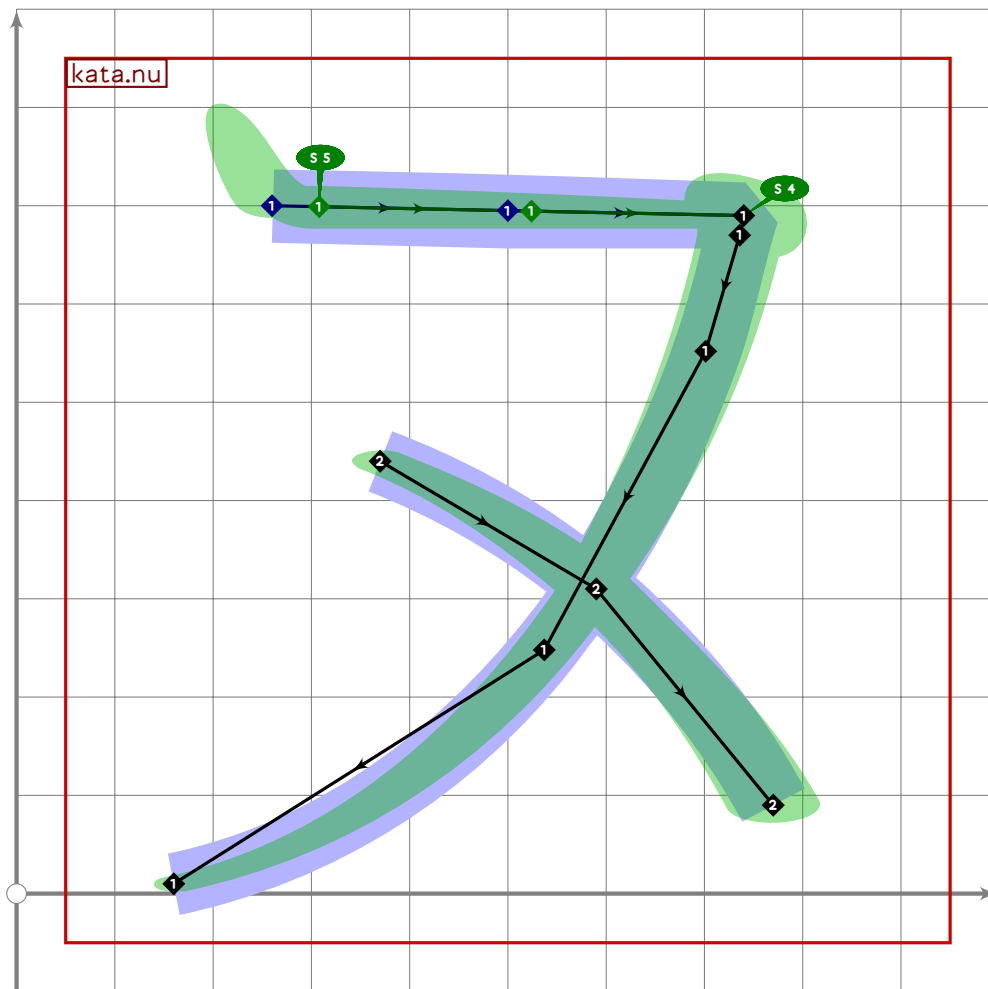


```

379
380 vardef kata.ni =
381   push_pbox_toexpand("kata.ni");
382
383   push_stroke((220,600)–(500,600)–(780,600),
384     (0,7,2,7)–(1,5,1,5)–(0,7,2,9));
385   set_boserif(0,0,5);
386   set_boserif(0,2,6);
387
388   push_stroke((110,140)–(500,140)–(890,140),
389     (0,7,2,7)–(1,5,1,5)–(0,7,2,9));
390   set_boserif(0,0,5);
391   set_boserif(0,2,6);
392   expand_pbox;
393 enddef;

```

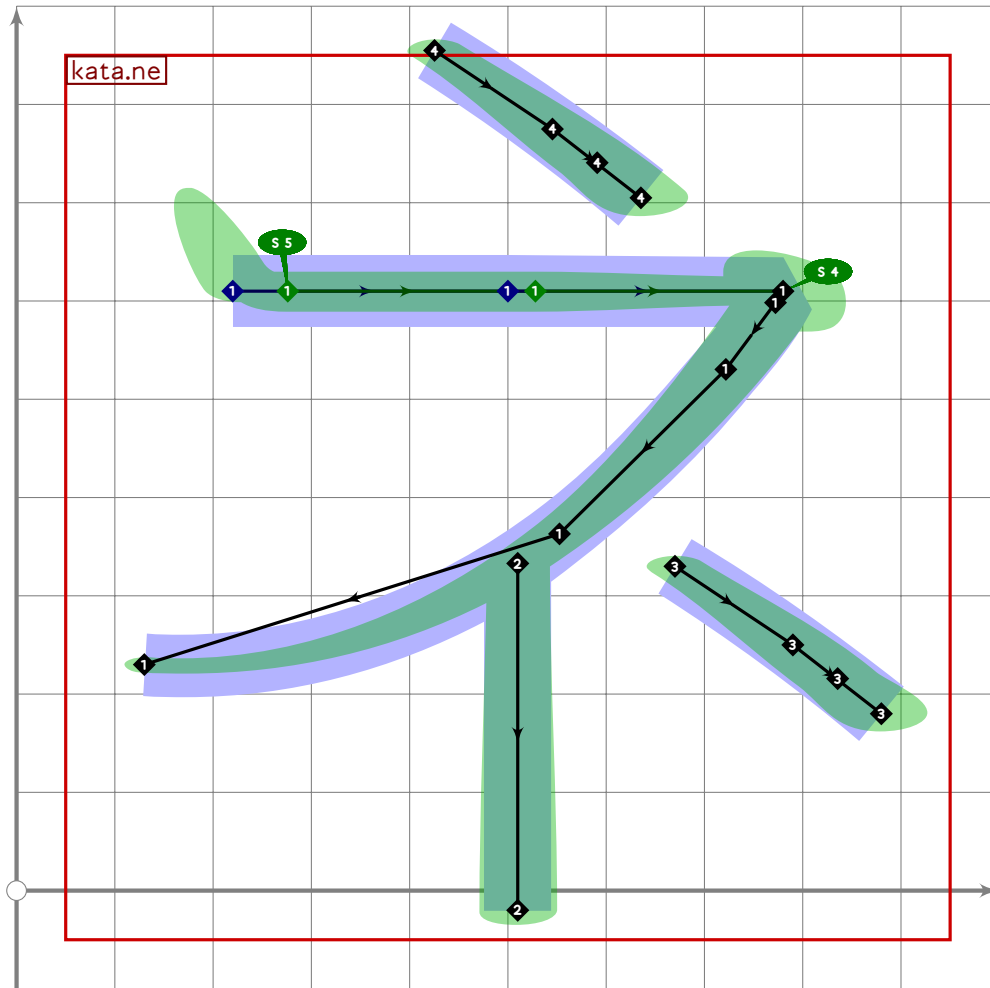
KATA



```

394
395 vardef kata.nu =
396   push_pbox__toexpand("kata.nu");
397
398   kata.fu_stroke((260,700),(740,690),(160,10));
399
400   push_stroke((370,440)..(590,310)..(770,90),
401     (1.3,1.3)–(1.6,1.6)–(1.8,1.8));
402   expand_pbox;
403 enddef;

```



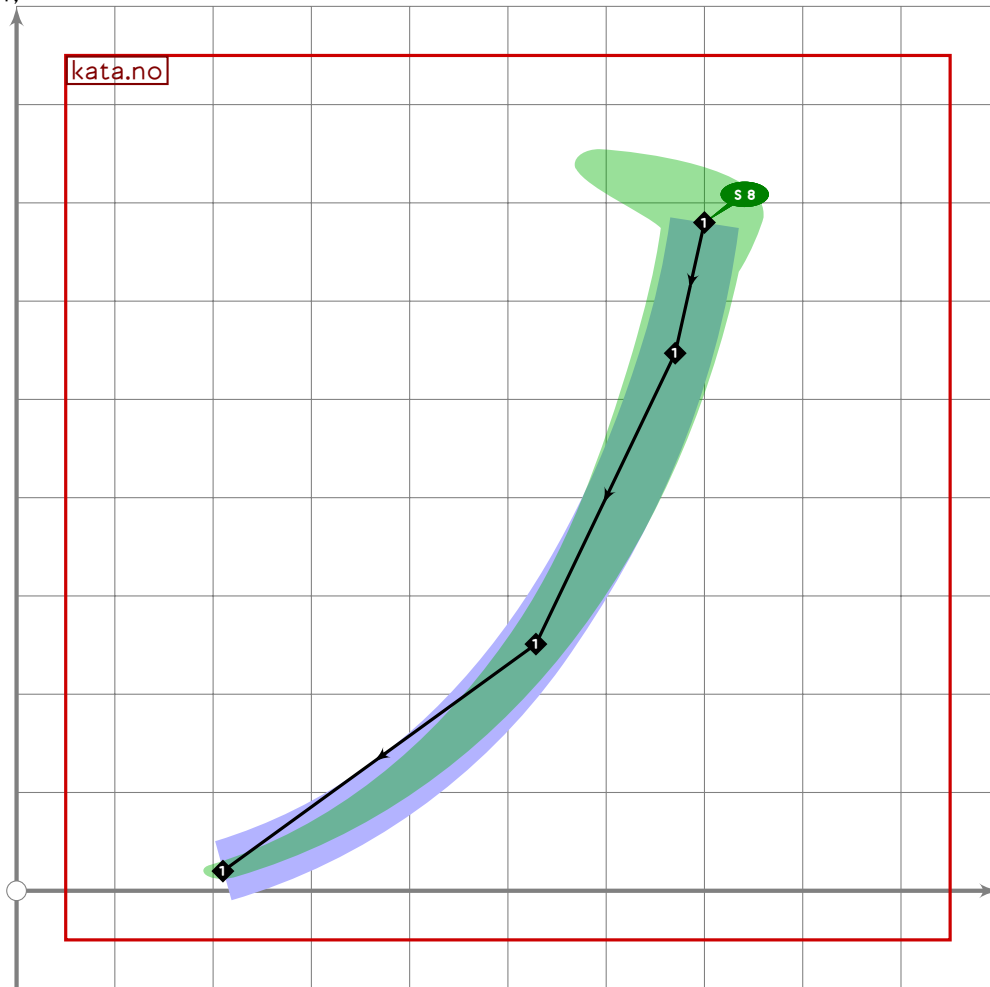
```

404
405 vardef kata.ne =
406   push_pbox__toexpand("kata.ne");
407
408   kata.fu_stroke((220,610),(780,610),(130,230));
409
410   push_stroke((((510,0)-(510,1000)) intersectionpoint
411     reverse get_stroke(0)-(510,-20),
412     (14,14)-(1.6,1.6));
413
414   push_stroke((670,330)..(790,250)..(880,180),
415     (1.3,1.3)-(1.6,1.6)-(1.8,1.8));
416
417   push_stroke(get_stroke(0) shifted ((510,800)-point 0.7 of get_stroke(0)),
418     get_stroke(0));
419   expand_pbox;
420 enddef;
421
422 vardef kata.no_stroke(expr ur,ll) =
423   push_stroke(insert_nodes(ur..tension 1.1..
424     (0.65[xpart ll,xpart ur],0.35[ypart ll,ypart ur])..{curl 1.2}ll)(0.3),

```



```
425 (1.7,1.7)-(1.7,1.7)-(1.4,1.4)-(1.1,1.1));
426 enddef;
```

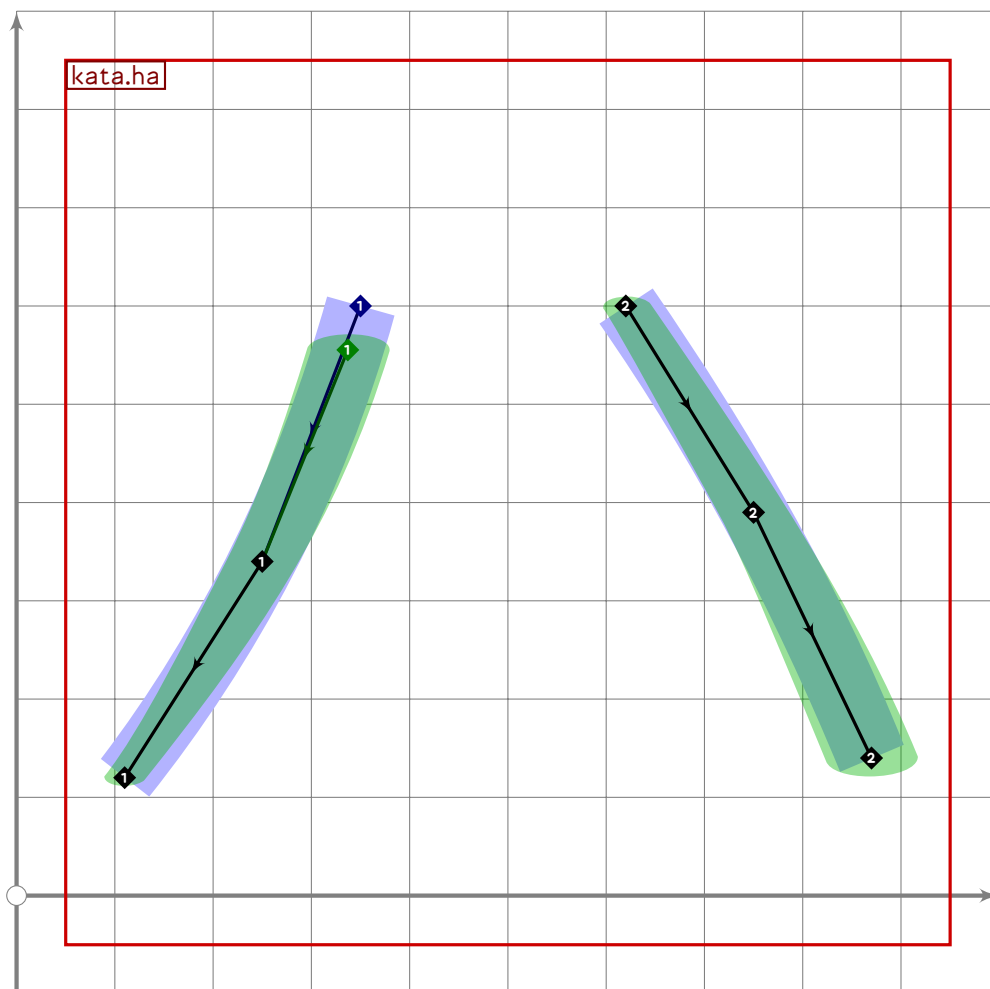


```
427
428 vardef kata.no =
429   push_pbox_toexpand("kata.no");
430
431   kata.no_stroke((700,680),(210,20));
432   set_boserif(0,0,8);
433   expand_pbox;
434 enddef;
435
```

KATA

Katakana Hahifuheho/Babibubebo/Papipupepo

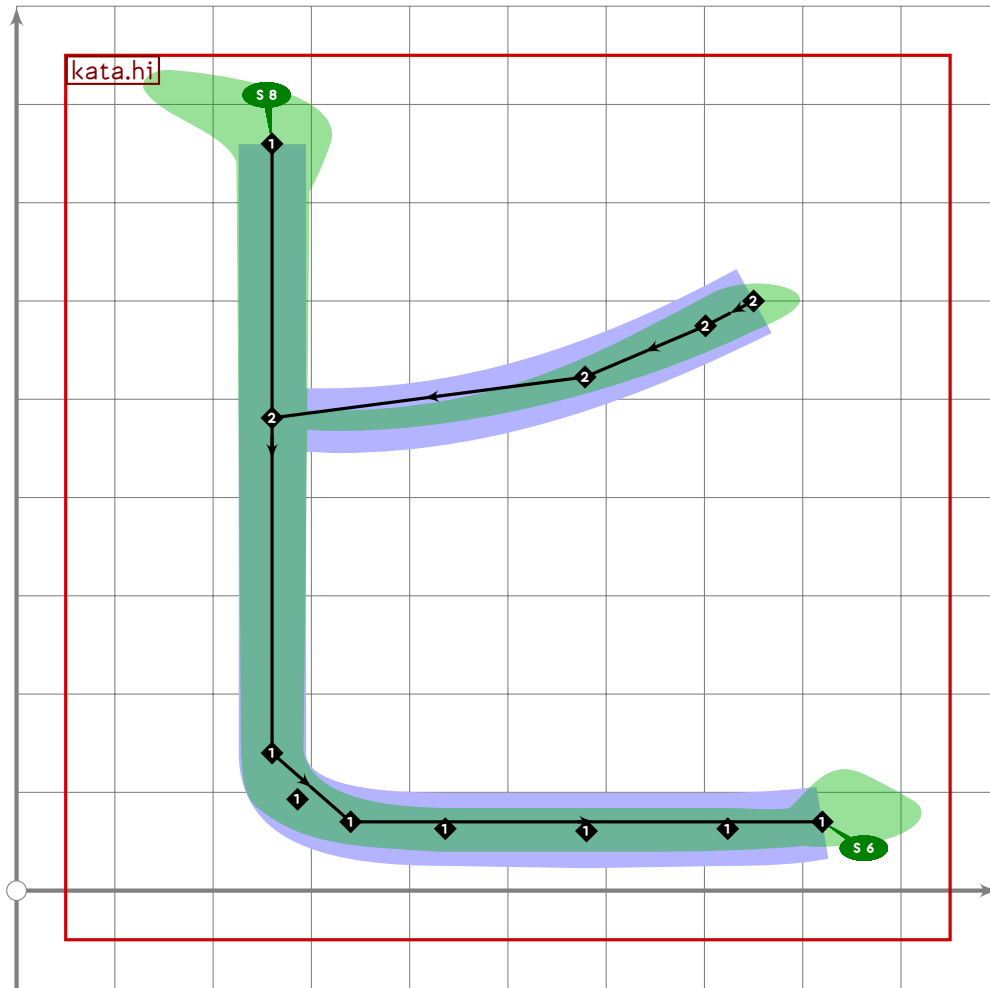
```
436 %%%%%%%%% KATAKANA HAHIFUHEHO/BABIBUBEBO/PAPIPUPEPO
```



```

437
438 vardef kata.ha =
439   push_pbox_toexpand("kata.ha");
440
441   push_stroke((350,600)..(250,340)..(110,120),
442     (0.7,2.7)-(1.5,1.5)-(1.1,1.1));
443   set_boserif(0,0,8);
444
445   push_stroke((620,600)..(750,390)..(870,140),
446     (1.2,1.2)-(1.5,1.5)-(1.8,1.8));
447   expand_pbox;
448 enddef;

```



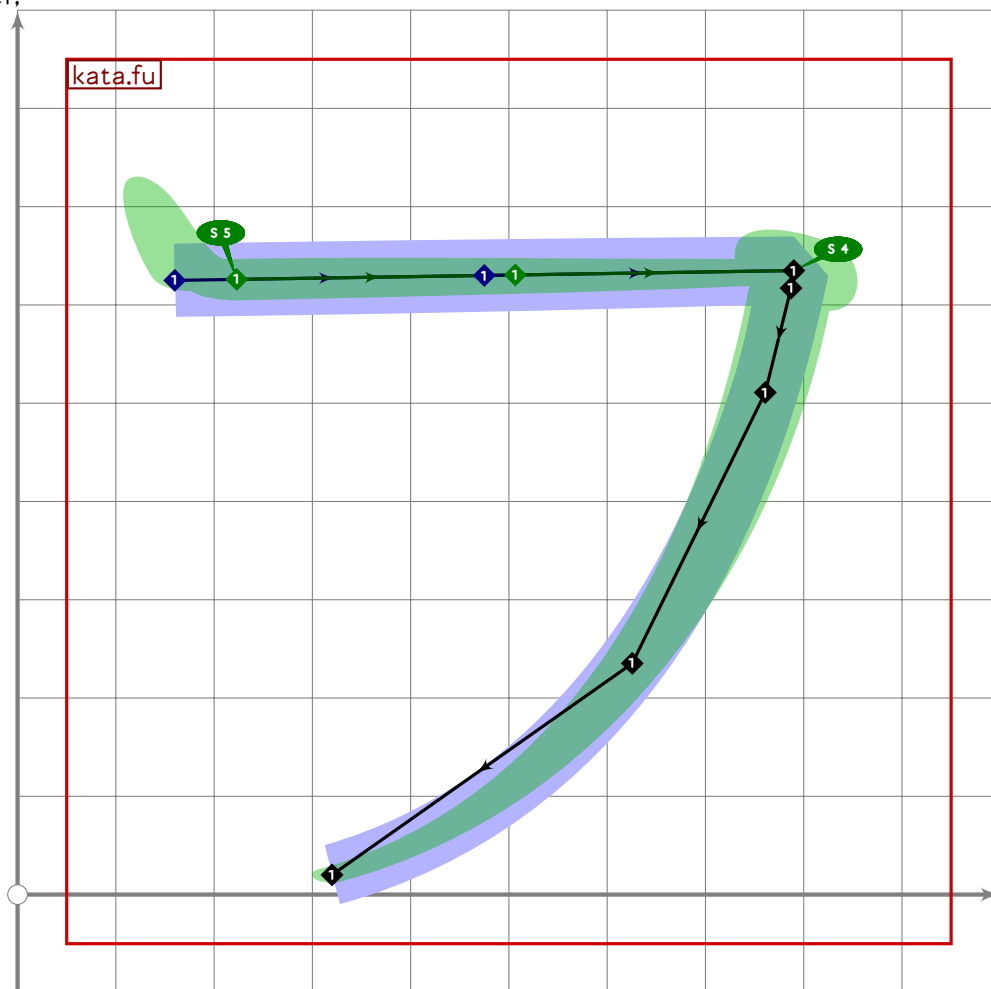
```

449
450 vardef kata.hi =
451   push_pbox__toexpand("kata.hi");
452
453   push_stroke((260,760)–(260,140){dir 274}..(340,70)..tension 2.1..(820,70),
454     (0.84,2.18)–(14,14)–(2.1,2.1)–(19,19));
455   set__boserif(0,0,8);
456   set__boserif(0,3,6);
457
458   kata.no__stroke((750,600),point 0.45 of get__strokep(0));
459   replace__strokeq(0)(oldq shifted (0.1,0.1));
460   expand__pbox;
461 enddef;
462
463 vardef kata.fu__stroke(expr ul,ur,ll) =
464   kata.no__stroke(ur,ll);
465   replace__strokep(0)(insert__nodes(((mincho*0.1)[ul,ur])–oldp)(0.5,1.15));
466   replace__strokeq(0)((2,2)–(19,19)–(1.5,1.5)–oldq);
467   set__botip(0,2,0);
468   set__boserif(0,0,5);
469   set__boserif(0,2,4);

```

U+30D5
tsuku.uni30D5

470 endif;



471

472 vardef kata.fu =

473 push_pbox_toexpand("kata.fu");

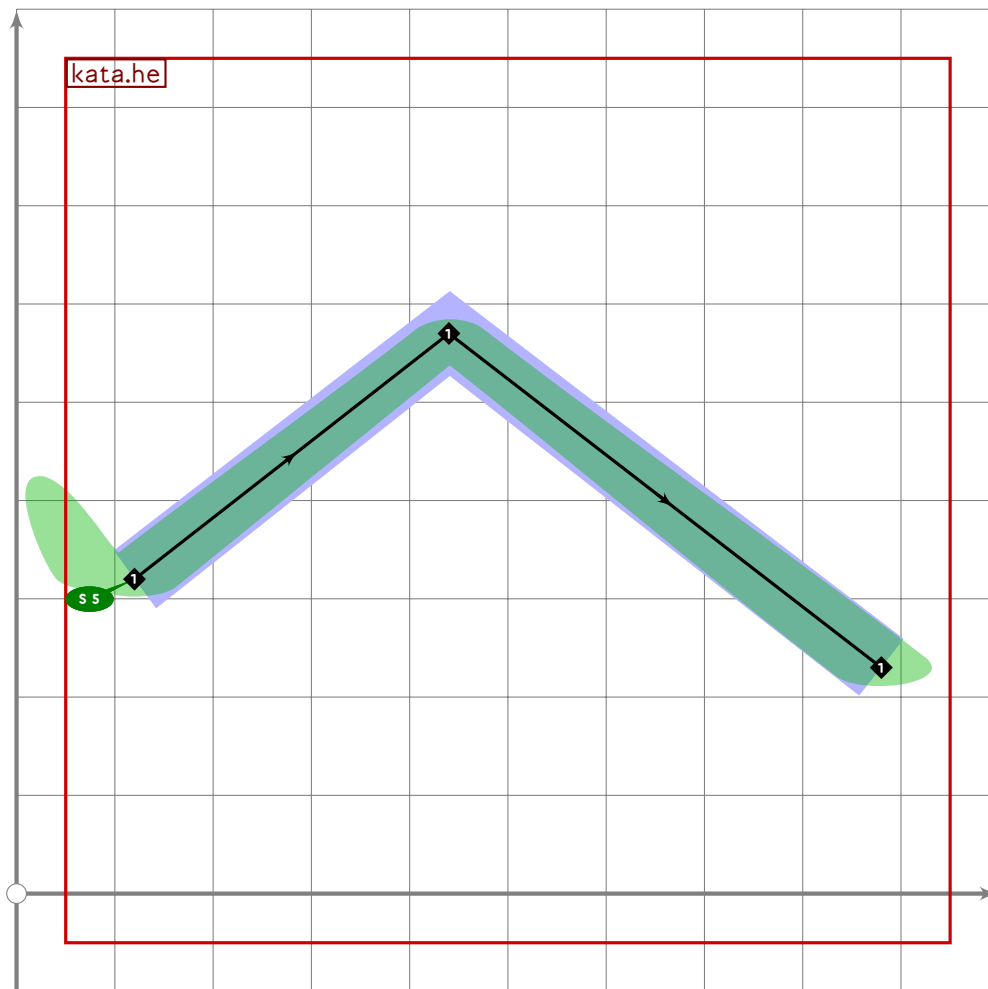
474

475 kata.fu_stroke((160,625),(790,635),(320,20));

476 expand_pbox;

477 endif;

KATA



```

478
479 vardef kata.he =
480   push_pbox__toexpand("kata.he");
481
482   push_stroke((120,320)–(440,570)–(880,230),
483     (1.8,1.8)–(1.5,1.5)–(1.9,1.9));
484   set_botip(0,1);
485   set_boserif(0,0,5);
486   expand_pbox;
487 enddef;
488
489 vardef kata.ho_centre(expr pta,ptb) =
490   push_stroke(begingroup
491     numeric x[],y[];
492     path mycirc,ripx,ripy;
493     mycirc:=fullcircle scaled 100 shifted ptb;
494     z1=(pta–ptb) intersectionpoint mycirc;
495     z2=ptb+(-200,40);
496     z3=0.85[z2,z1];
497     ripx:=pta{down}..tension 1.6..z3.{curl 0}z2;
498     ripx:=pta{down}...(point 0.95 of ripx)..z3.{curl 0}z2;

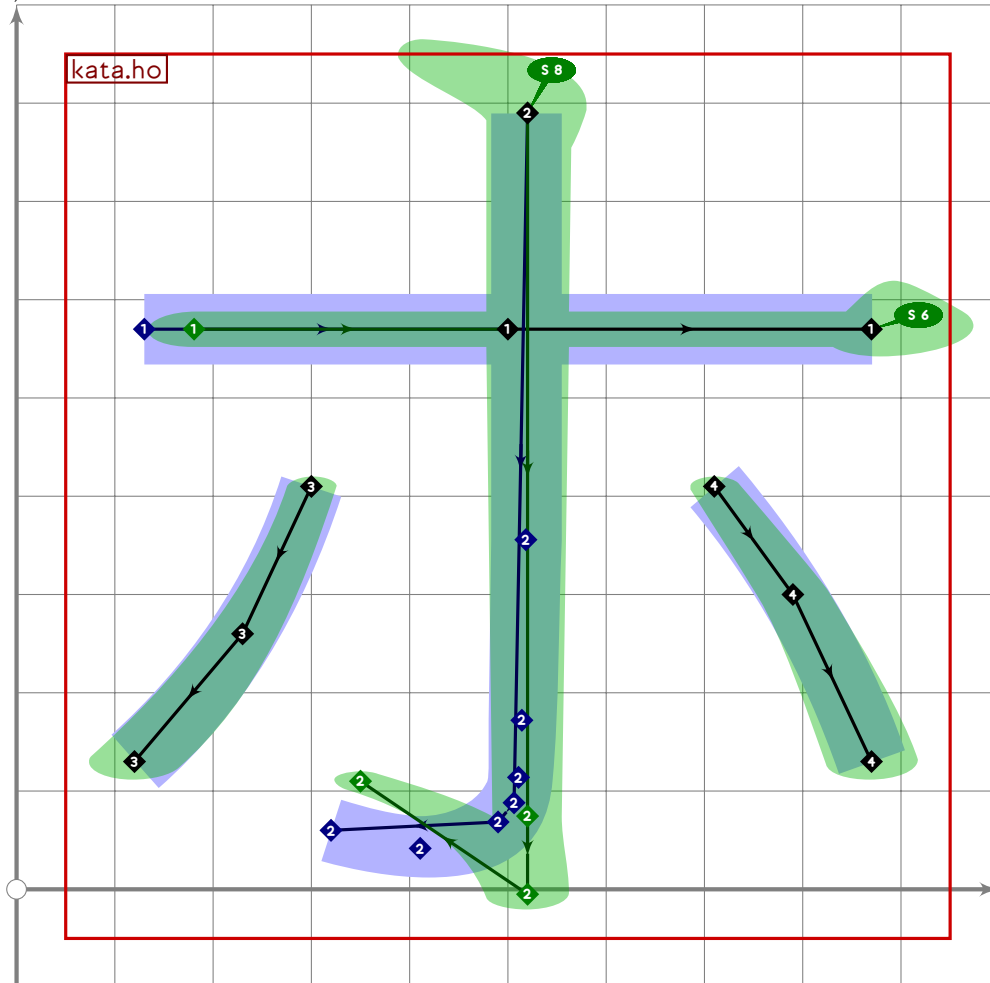
```

U+30DB
tsuku.uni30DB

```

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

```



```

509
510   vardef kata.ho =
511     push_pbox_toexpand("kata.ho");
512
513     push_stroke((130,570)-(500,570)-(870,570),
514       (0.68,2.92)-(1.8,1.8)-(1.9,1.9));
515     set_boserif(0,0,5);
516     set_boserif(0,2,6);
517
518     kata.ho_centre((520,790),(520,20));
519

```

KATA

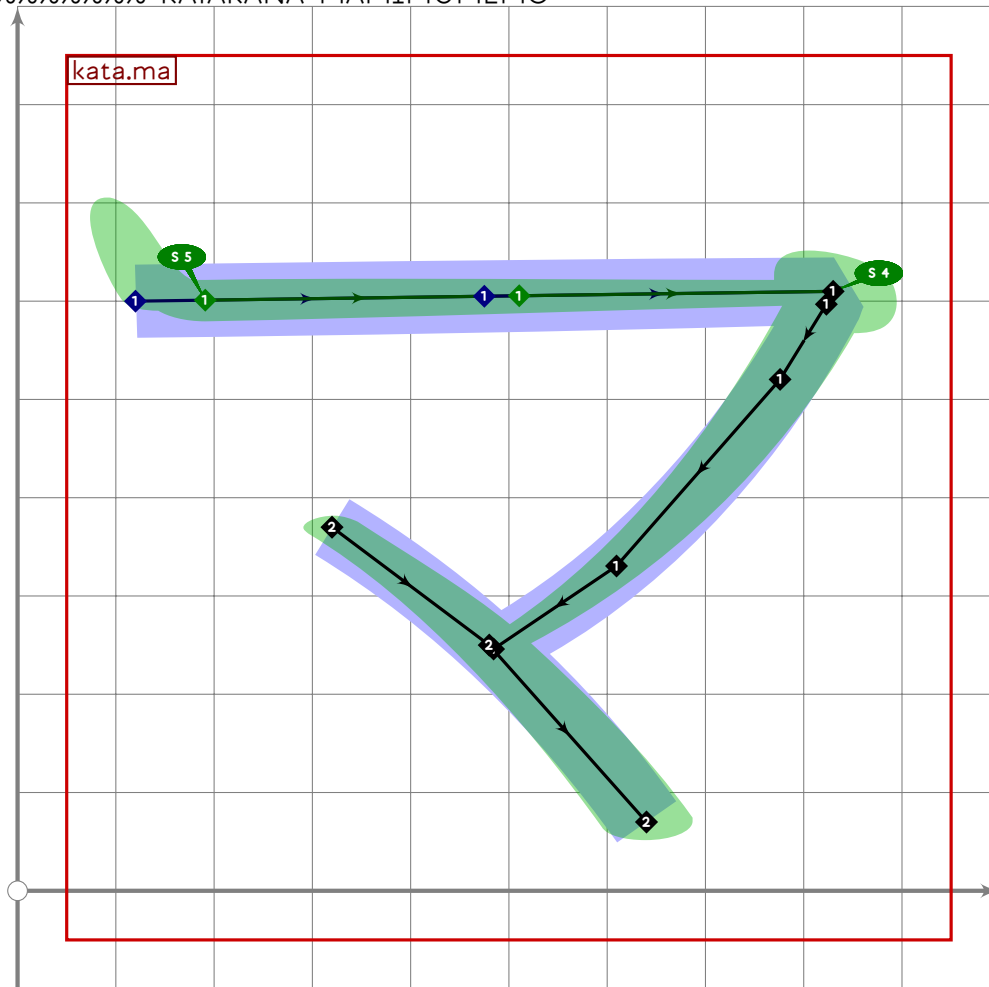
```

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

```

Katakana Mamimumemo

528 %%%%%%%%% KATAKANA MAMIMUMEMO



```

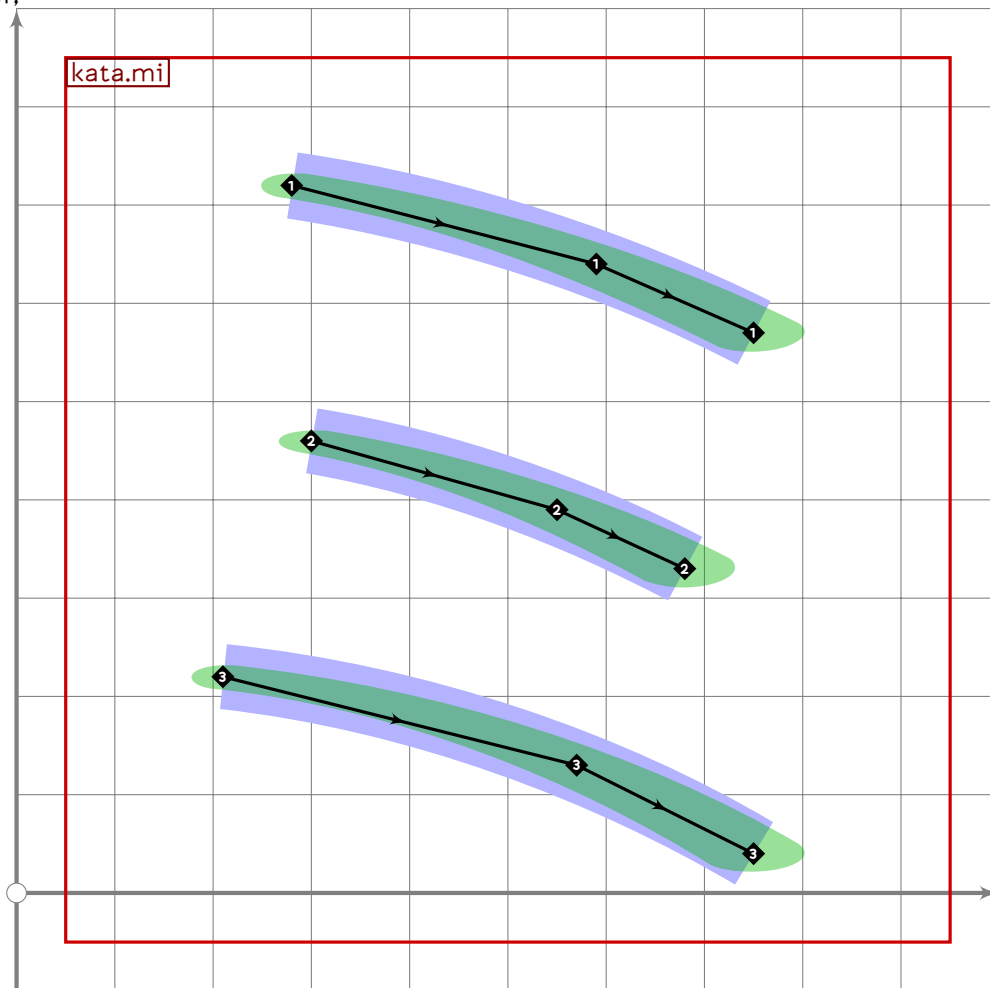
529
530 vardef kata.ma =
531   push_pbox_toexpand("kata.ma");
532
533   kata.fu_stroke((120,600),(830,610),(200,180));
534
535   push_stroke((320,370)..(480,250)..(640,70),
536     (1.3,1.3)–(1.6,1.6)–(1.8,1.8));
537   replace_strokep(-1)(subpath (0,xpart (oldp intersectiontimes

```

KATA

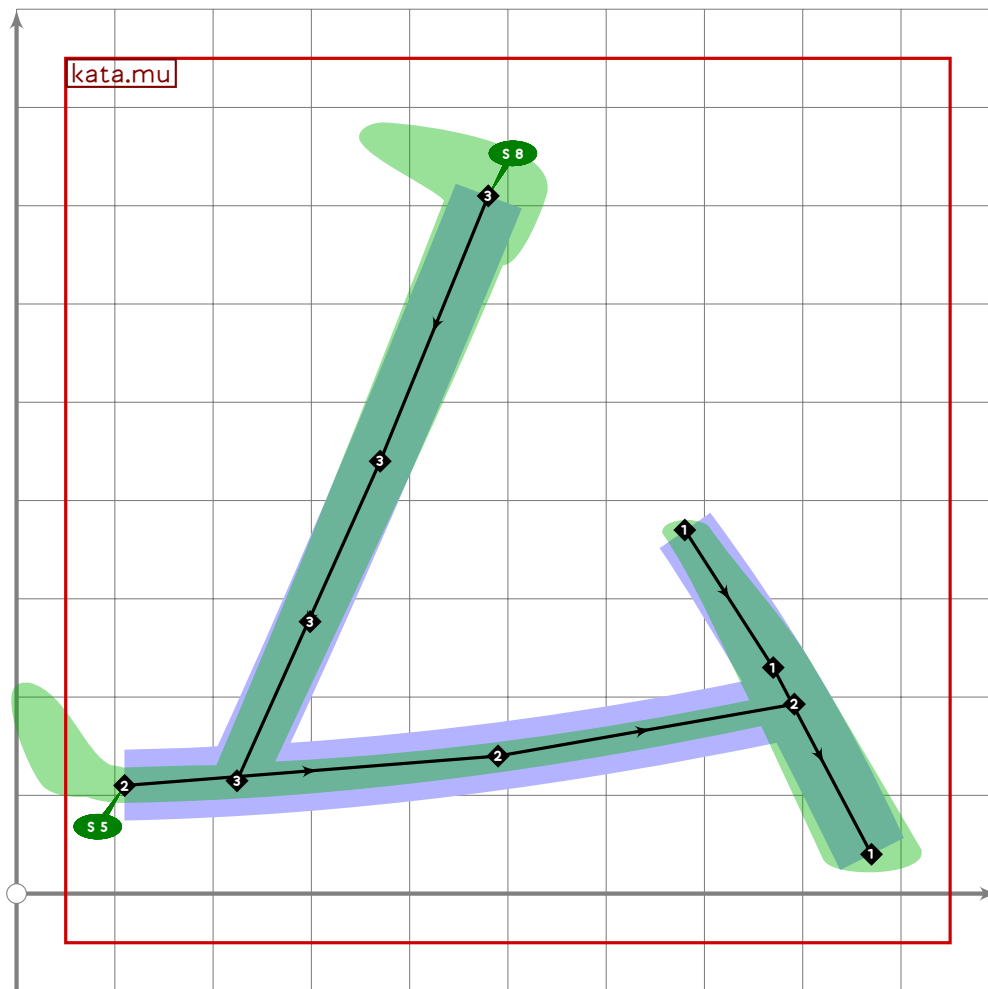
U+30DF
tsuku.uni30DF

```
538   get_strokep(0))) of oldp);
539   expand_pbox;
540 enddef;
```



```
541
542 vardef kata.mi =
543   push_pbox_toexpand("kata.mi");
544
545   push_stroke((280,720)..(590,640)..(750,570),
546     (1,1,4)-(1,7,1)-(1,9,1));
547
548   push_stroke((300,460)..(550,390)..(680,330),
549     (1,1,4)-(1,7,1)-(1,9,1));
550
551   push_stroke((210,220)..(570,130)..(750,40),
552     (1,1,4)-(1,7,1)-(1,9,1));
553   expand_pbox;
554 enddef;
```

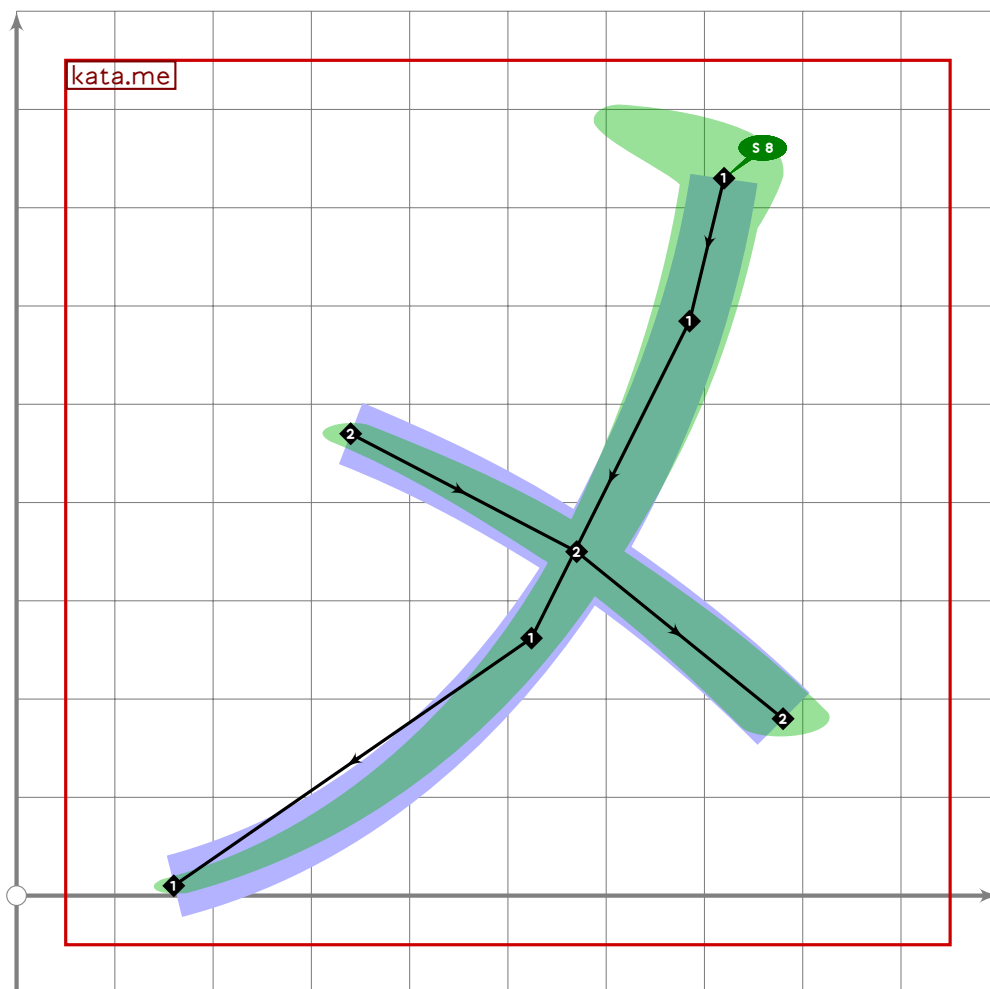
KATA



```

555
556 vardef kata.mu =
557   push_pbox_toexpand("kata.mu");
558
559   push_stroke((680,370)..(770,230)..(870,40),
560     (1,2,1,2)-(1,6,1,6)-(1,9,1,9));
561
562   push_stroke((110,110)..(490,140)..(point 1.2 of get_stroke(0)),
563     (1,8,1,8)-(1,6,1,6)-(1,4,1,4));
564   set_boserif(0,0,5);
565
566   push_stroke((480,710)..(370,440)..(point 0.3 of get_stroke(0)),
567     (1,7,1,7)-(1,5,1,5)-(1,3,1,3));
568   set_boserif(0,0,8);
569   expand_pbox;
570 enddef;

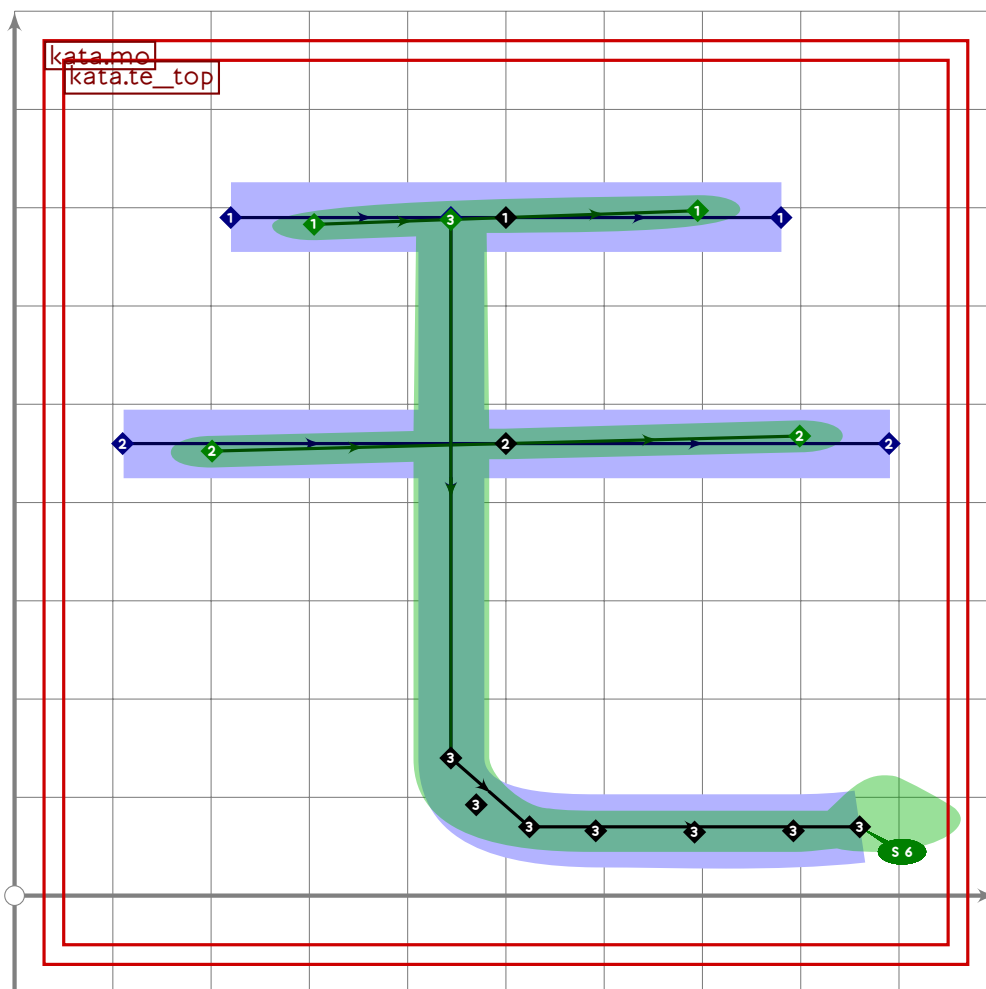
```



```

571
572 vardef kata.me =
573   push_pbox__toexpand("kata.me");
574
575   kata.no_stroke((720,730),(160,10));
576   set__boserif(0,0,8);
577
578   push_stroke((340,470)..(570,350)..(780,180),
579     (1.3,1.3)–(1.6,1.6)–(1.8,1.8));
580   expand_pbox;
581 enddef;

```



```

582
583 vardef kata.mo =
584   push_pbox__toexpand("kata.mo");
585
586   kata.te_top;
587
588   push_stroke((point 0.8 of get_stroke(-1))-
589     (xpart point 0.8 of get_stroke(-1),140){dir 274}..
590     (80+xpart point 0.8 of get_stroke(-1),70)..tension 21..(860,70),
591     (1.5,1.5)-(1.6,1.6)-(2,2)-(1.9,1.9));
592   set_boserif(0,3,6);
593   expand_pbox;
594 enddef;
595

```

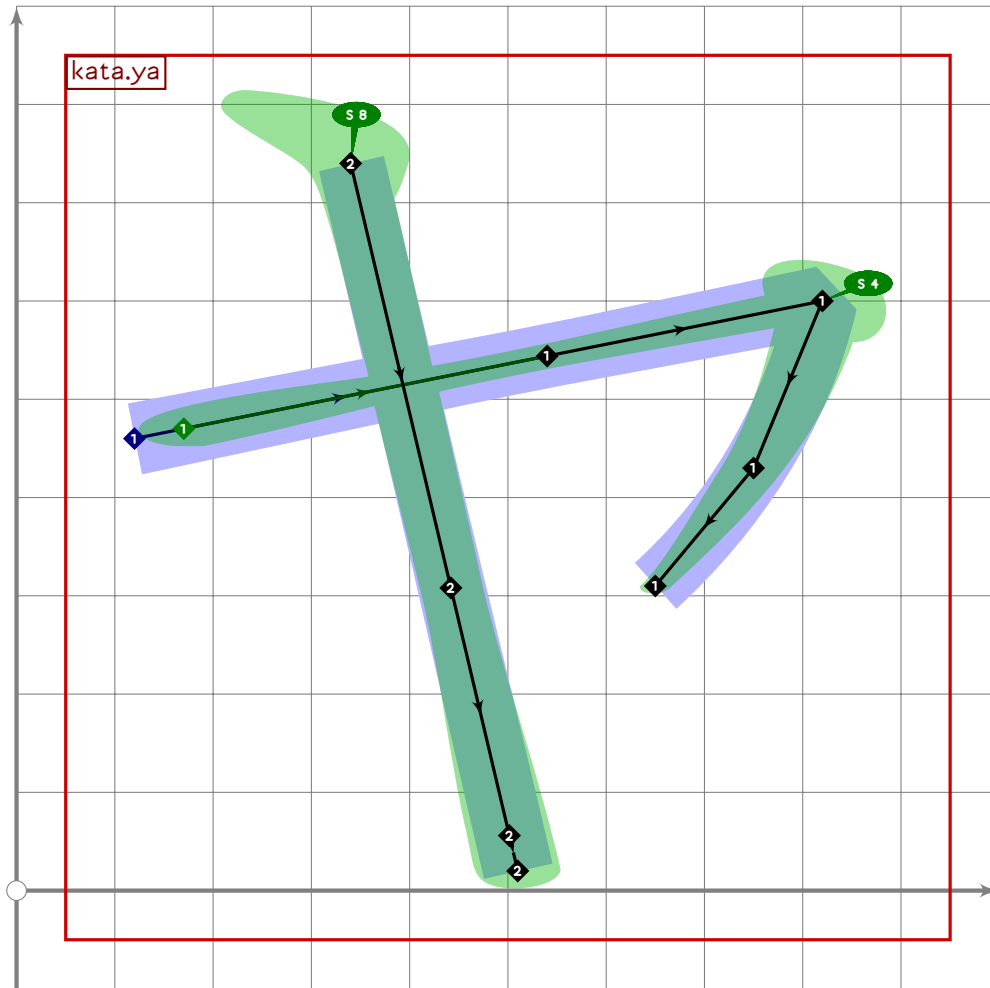
KATA

Katakana Yayuyo

```

596 %%%%%%%%% KATAKANA YAYUYO

```

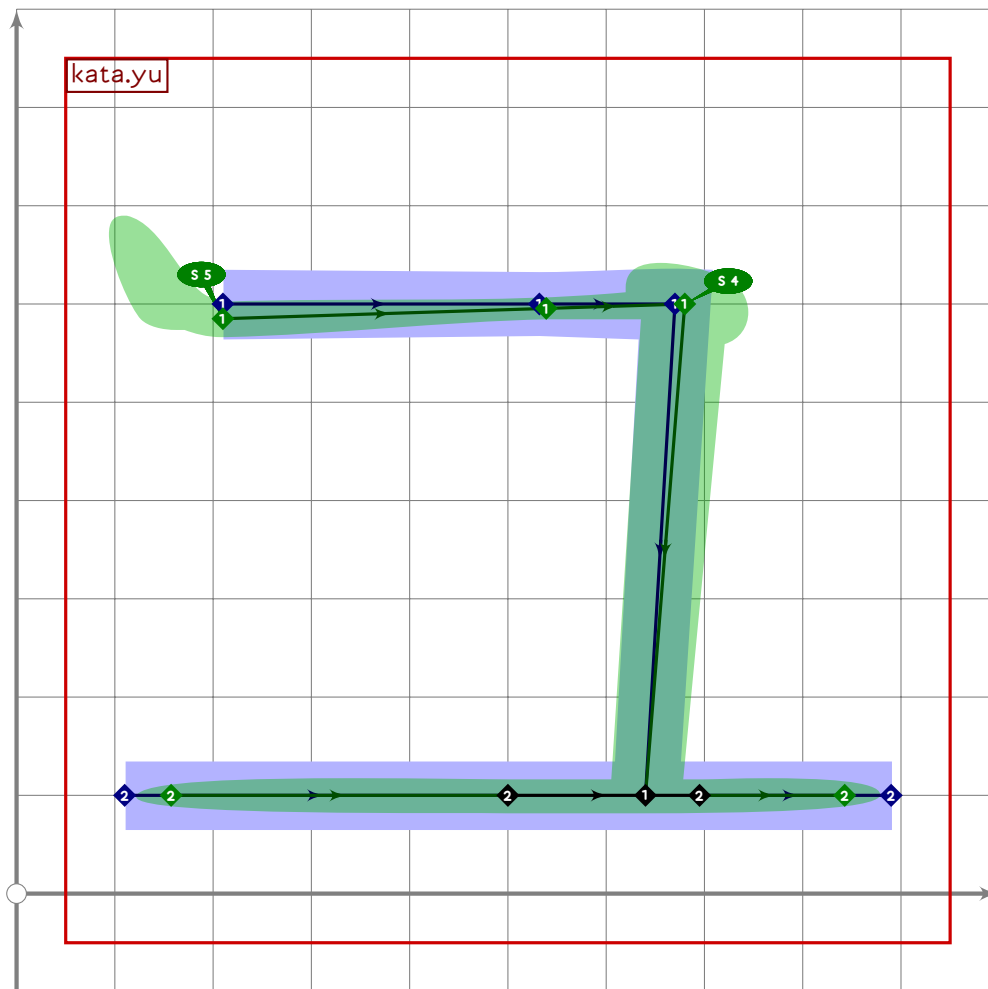


```

597
598 vardef kata.ya =
599   push_pbox_toexpand("kata.ya");
600
601   push_stroke((120,460)–(820,600)..(750,430)..(650,310),
602     (0.77,2.9)–(1.3,1.3)–(1.7,1.7)–(1.4,1.4)–(1,1));
603   replace_strokep(0)(insert_nodes(oldp)(0.6));
604   set_botip(0,2,0);
605   set_boserif(0,0,5);
606   set_boserif(0,2,4);
607
608   push_stroke((340,740)–(510,20),
609     (1.5,1.5)–(1.4,1.4)–(1.7,1.7)–(1.7,1.7));
610   replace_strokep(0)(insert_nodes(oldp)(0.6,0.95));
611   set_boserif(0,0,8);
612   expand_pbox;
613 enddef;

```

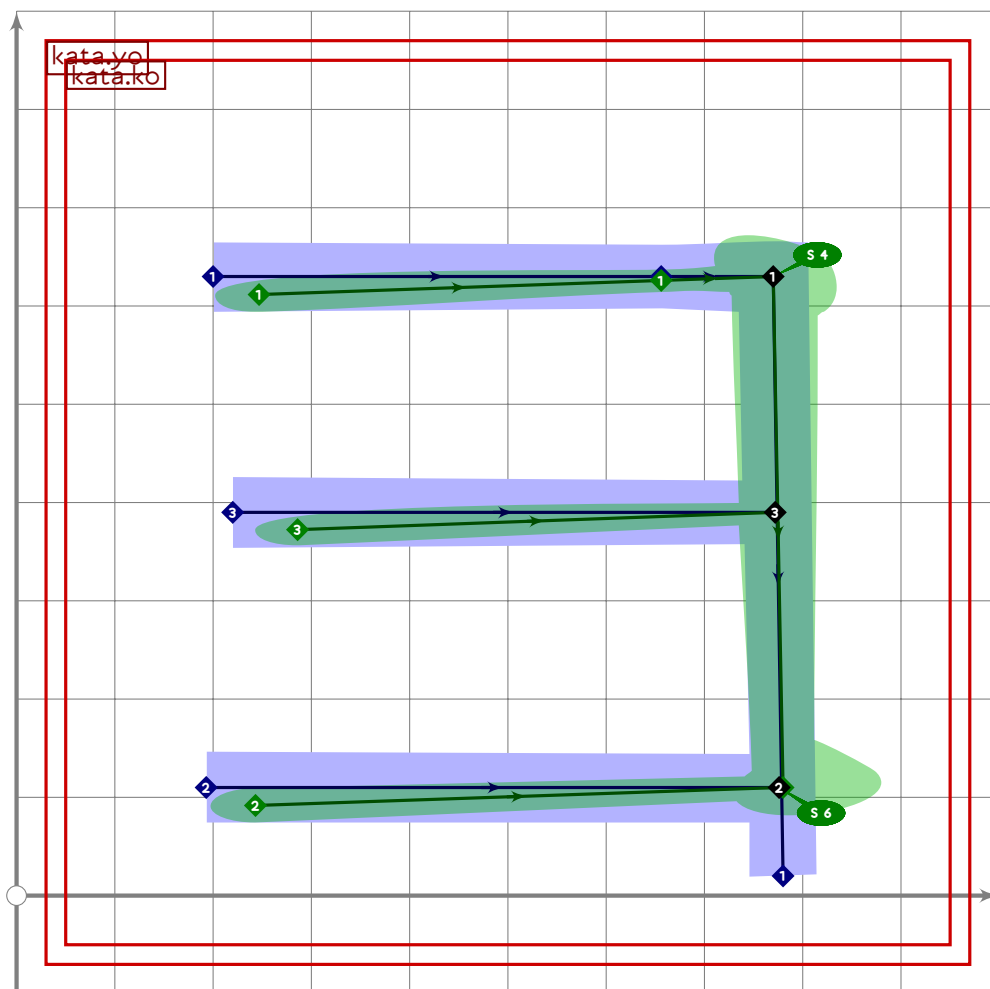
KATA



```

614
615 vardef kata.yu =
616   push_pbox_toexpand("kata.yu");
617
618   push_stroke((210,600-15*mincho)-(670+10*mincho,600)-(640,100),
619     (1.8,1.8)-(1.2,1.2)-(1.7,1.7)-(1.5,1.5));
620   replace_strokep(0)(insert_nodes(oldp)(0.7));
621   set_botip(0,2,1);
622   set_boserif(0,0,5);
623   set_boserif(0,2,4);
624
625   push_stroke((110,100)-(500,100)-(890,100),
626     (0.7,2.2)-(1.8,1.8)-(0.7,2.2));
627   % hack around FontAnvil cleanup bug
628   replace_strokep(0)(insert_nodes(oldp)(1.5));
629   replace_strokeq(0)(insert_nodes(oldq)(1.5));
630   set_boserif(0,0,5);
631   set_boserif(0,3,6);
632   expand_pbox;
633 enddef;

```



```

634
635 vardef kata.yo =
636   push_pbox__toexpand("kata.yo");
637
638   kata.ko;
639
640   push_stroke((220,390-20*mincho)-(772,390),
641     (0.77,2.7)-(1.3,1.3));
642   set_boserif(0,0,5);
643   expand_pbox;
644 enddef;
645

```

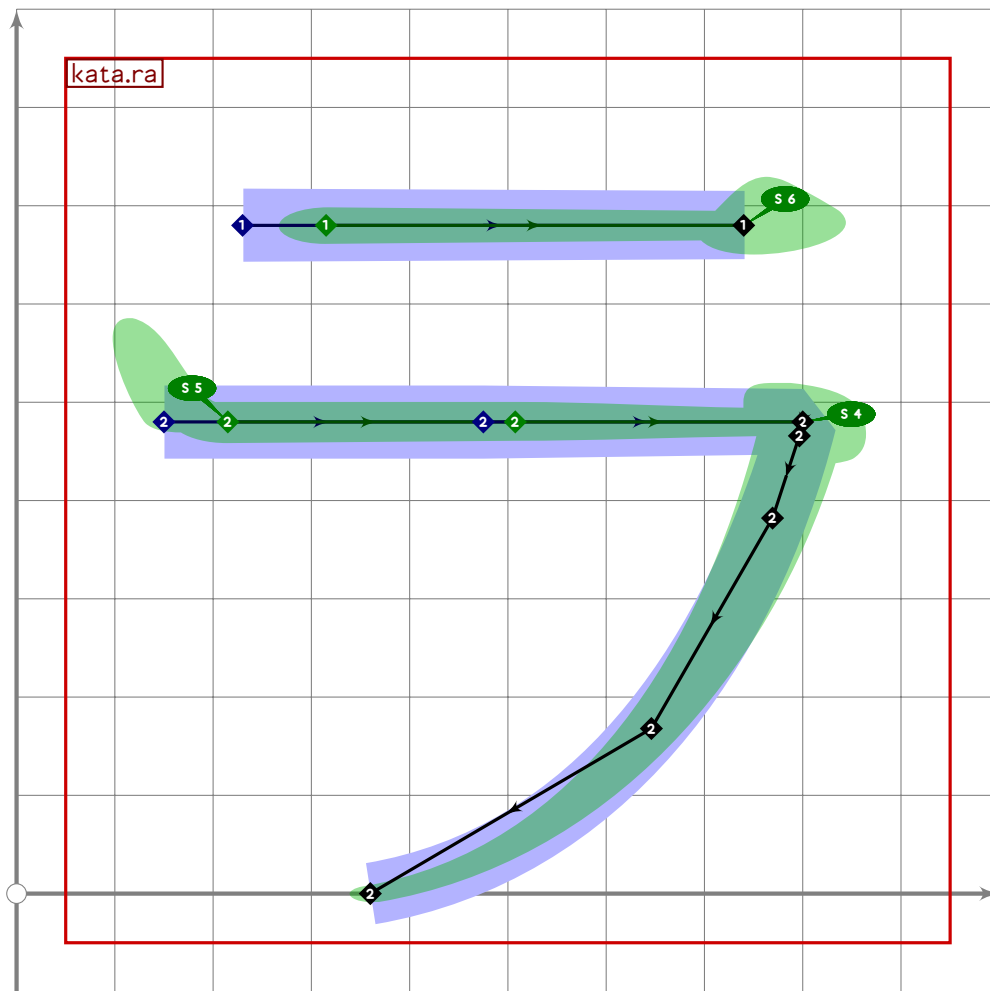
KATA

Katakana Rarirurero

```

646 %%%%%%%%% KATAKANA RARIRURERO

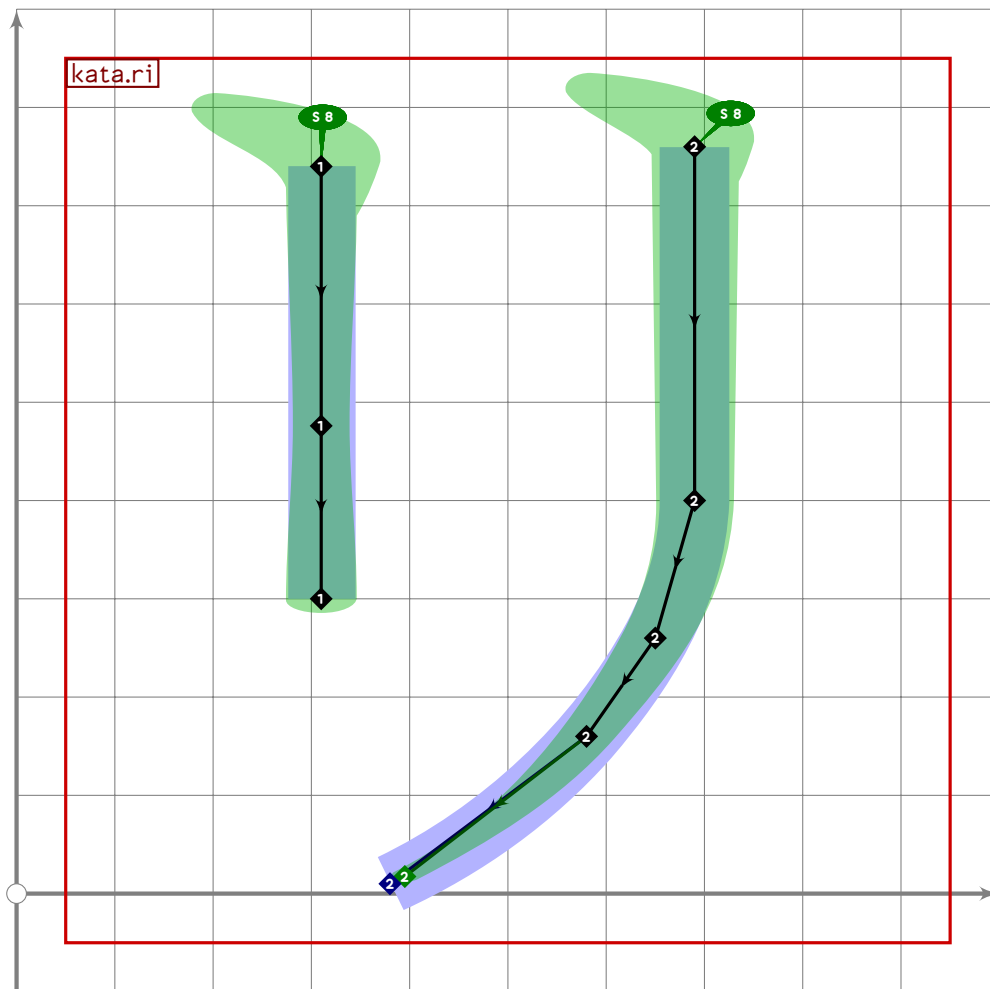
```



```

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

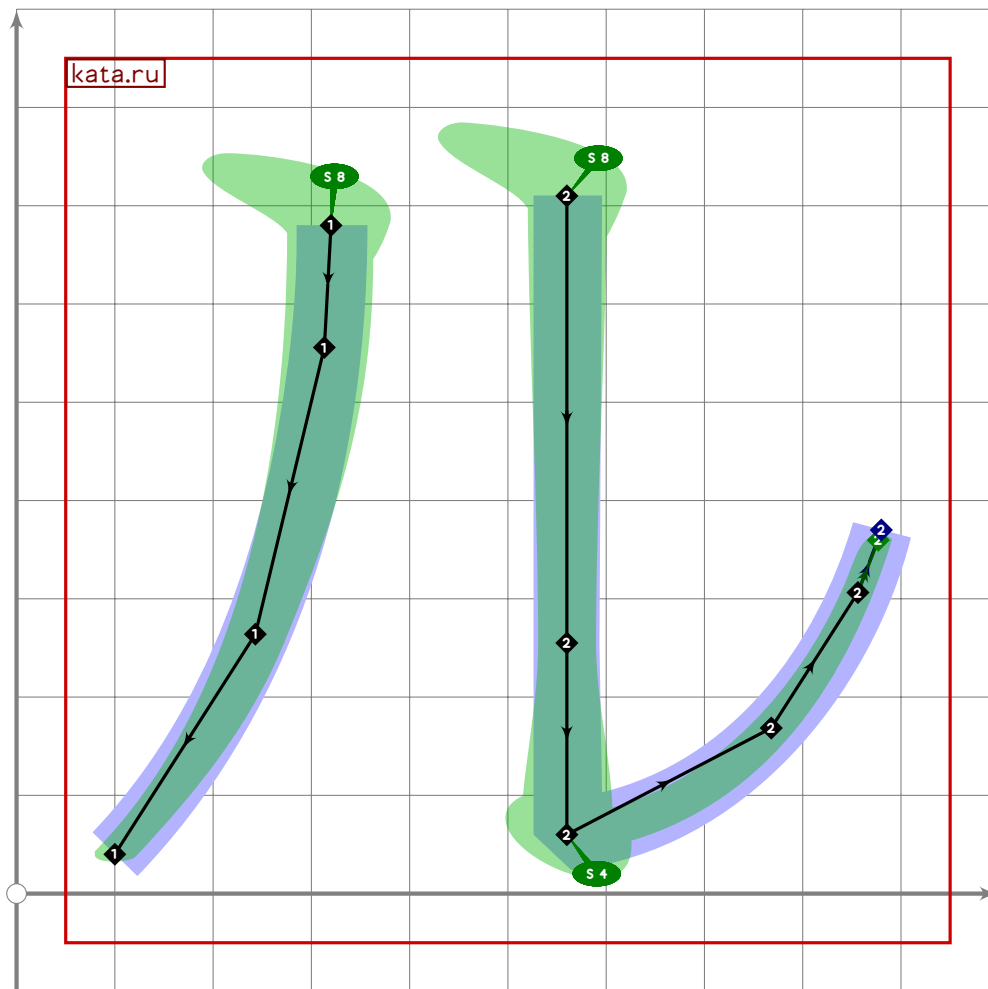
```



```

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

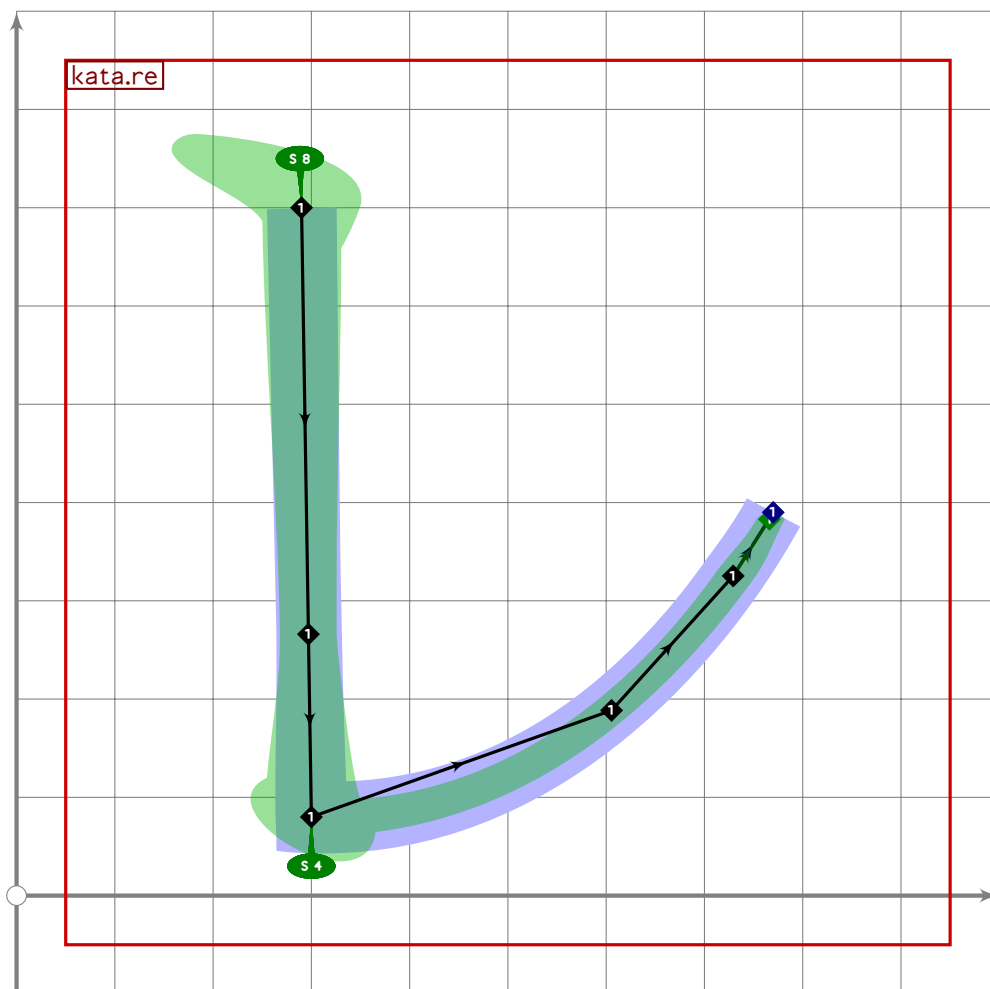
```

```

673
674 vardef kata.ru =
675   push_pbox_toexpand("kata.ru");
676
677   kata.no_stroke((320,680),(100,40));
678   set_boserif(0,0,8);
679
680   kata.no_stroke((880,370),(560,60));
681   replace_strokep(0)(insert_nodes((560,710)-reverse oldp)(0.7));
682   replace_strokeq(0)((1,6,1,6)-(1,3,1,3)-(1,8,1,8)-
683     (1,2,1,2)-(1,1)-(0,8,1));
684   set_botip(0,2,0);
685   set_boserif(0,0,8);
686   set_boserif(0,2,4);
687   expand_pbox;
688 enddef;

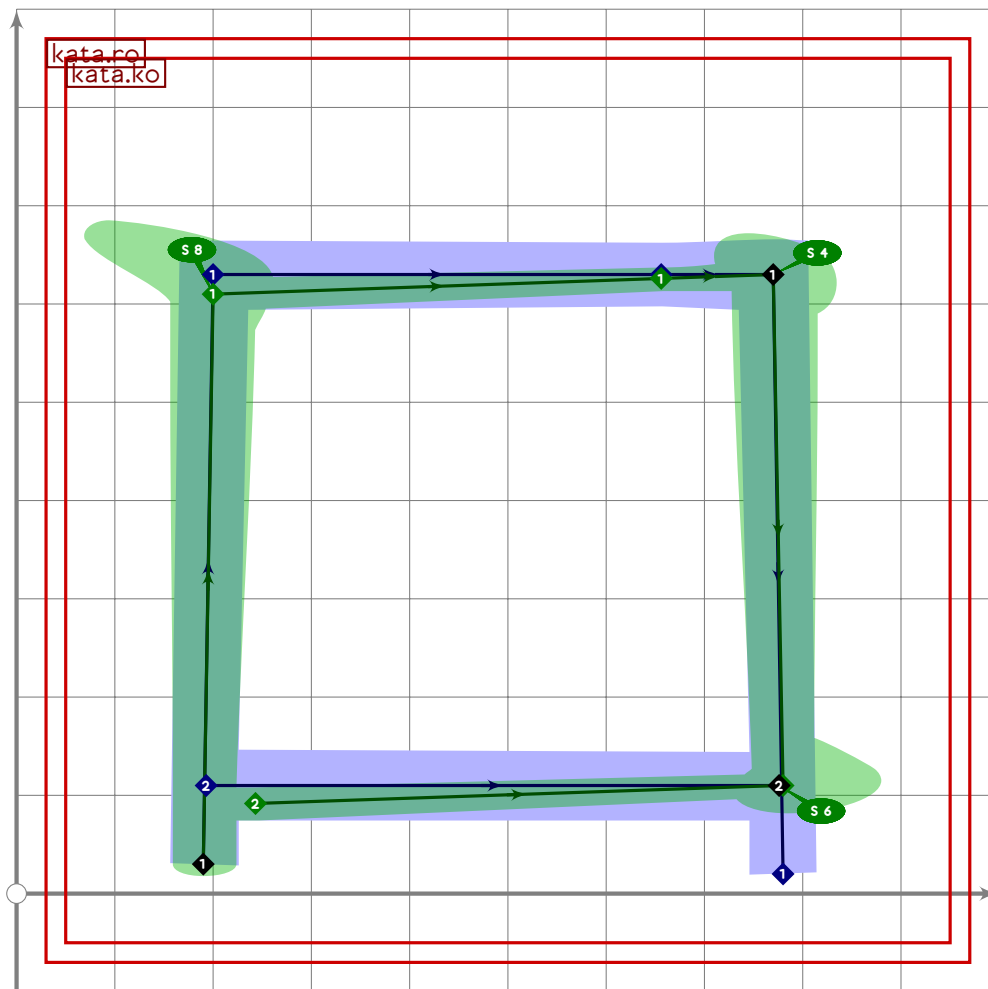
```



```

689
690 vardef kata.re =
691   push_pbox__toexpand("kata.re");
692
693   kata.no_stroke((770,390),(300,80));
694   replace_strokep(0)(insert_nodes((290,700)-reverse oldp)(0.7));
695   replace_strokeq(0)((1.6,1.6)-(1.3,1.3)-(1.8,1.8)-
696     (1.2,1.2)-(1.1,1.1)-(0.8,1));
697   set_botip(0,2,1);
698   set_boserif(0,0,8);
699   set_boserif(0,2,4);
700   expand_pbox;
701 enddef;

```



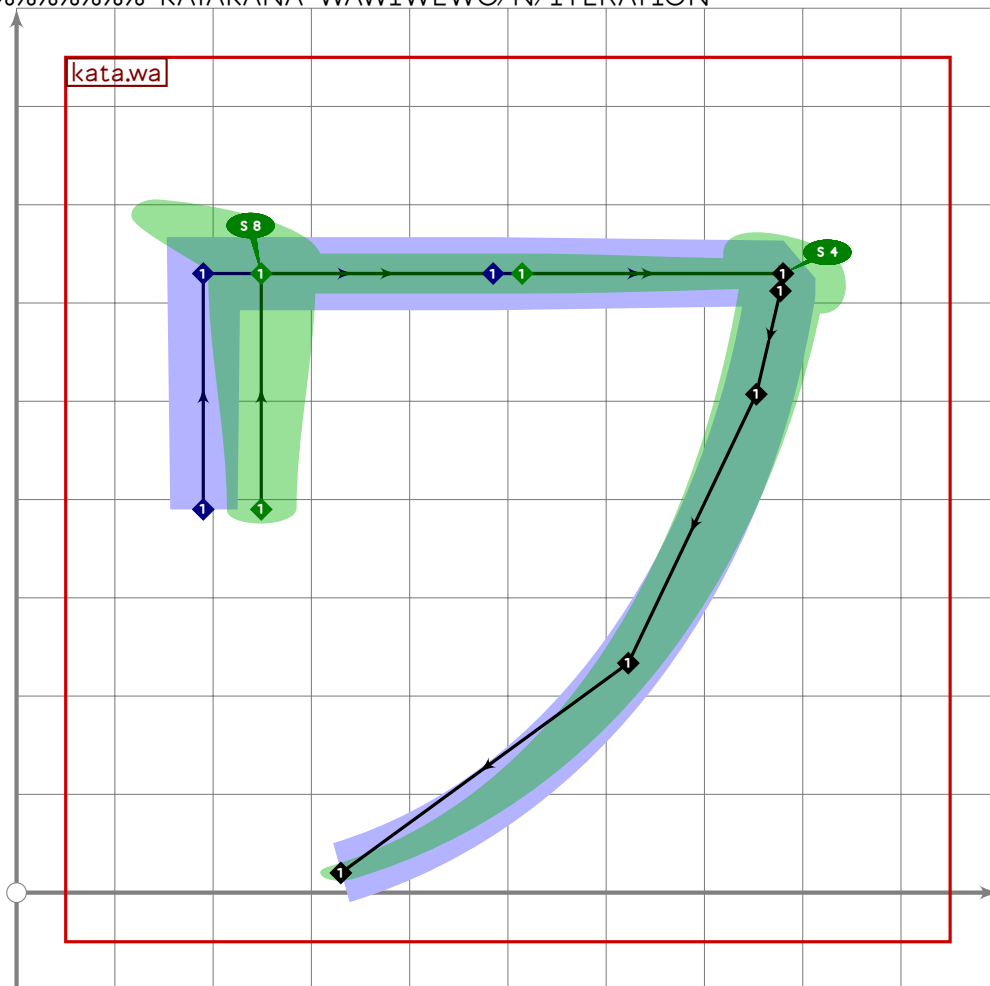
```

702
703 vardef kata.ro =
704   push_pbox__toexpand("kata.ro");
705
706   kata.ko;
707
708   replace_strokep(-1)((190,30)-oldp);
709   replace_strokeq(-1)((14,1,4)-(1,7,1,7)-(subpath (1,infinity) of oldq));
710   set_botip(-1,1,1);
711   set_botip(-1,2,whatever);
712   set_botip(-1,3,1);
713   set_boserif(-1,0,whatever);
714   set_boserif(-1,1,8);
715   set_boserif(-1,2,whatever);
716   set_boserif(-1,3,4);
717   set_boserif(0,0,whatever);
718   expand_pbox;
719 enddef;
720

```

Katakana Wawiwewo/N/Iteration

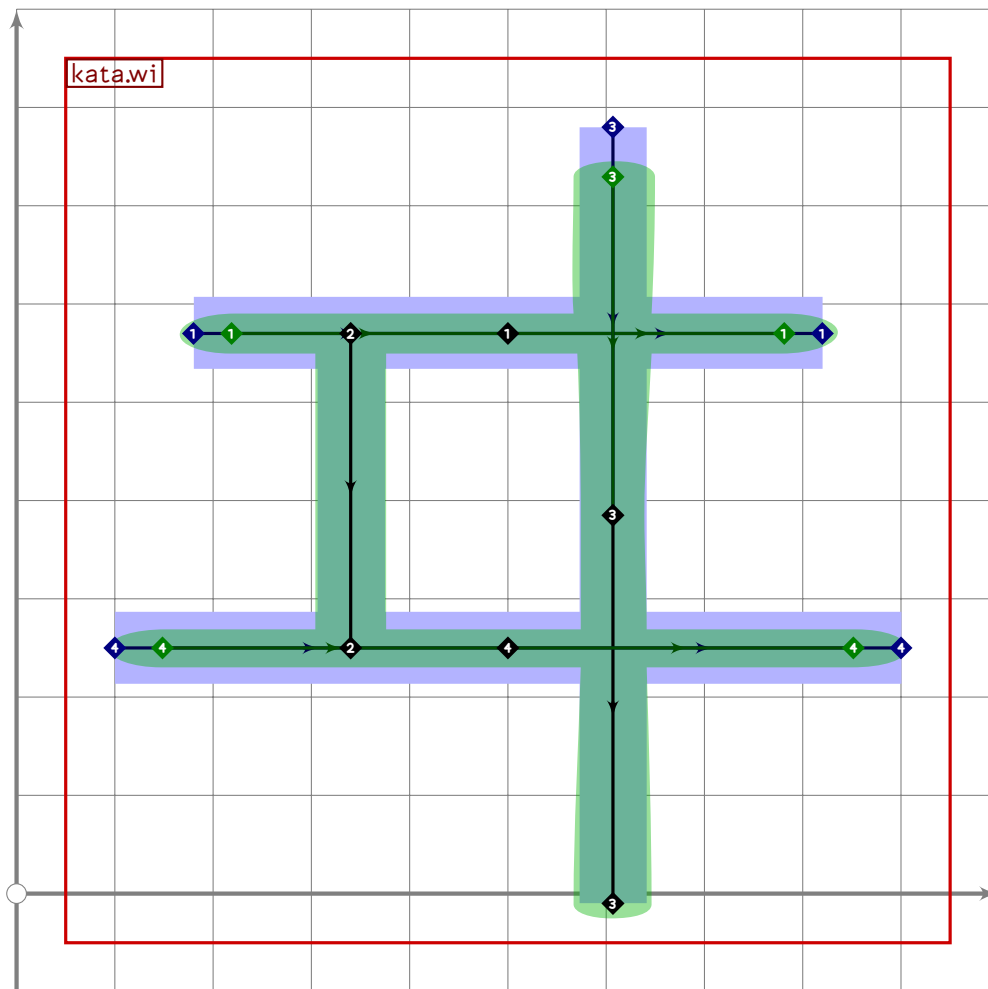
721 %%%%%%%%% KATAKANA WAWIWEWO/N/ITERATION



```

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

```



```

737
738 vardef kata.wi =
739   push_pbox_toexpand("kata.wi");
740
741   x1=100;
742   x2=180;
743   x3=0.25[x2,x5];
744   x4=0.667[x2,x5];
745   (x5+x2)/2=(x1+x6)/2=500;
746   y1=-10;
747   y2=250;
748   y3=570;
749   y4=780;
750   push_stroke((x2,y3)-(x5,y3),
751     (0.7,3.3)-(1.8,1.8)-(0.7,3.3));
752   replace_strokep(0)(insert_nodes(oldp)(0.5));
753   set_boserif(0,0,5);
754   set_boserif(0,2,6);
755
756   push_stroke((x3,y3)-(x3,y2),
757     (1.5,1.5)-(1.5,1.5));

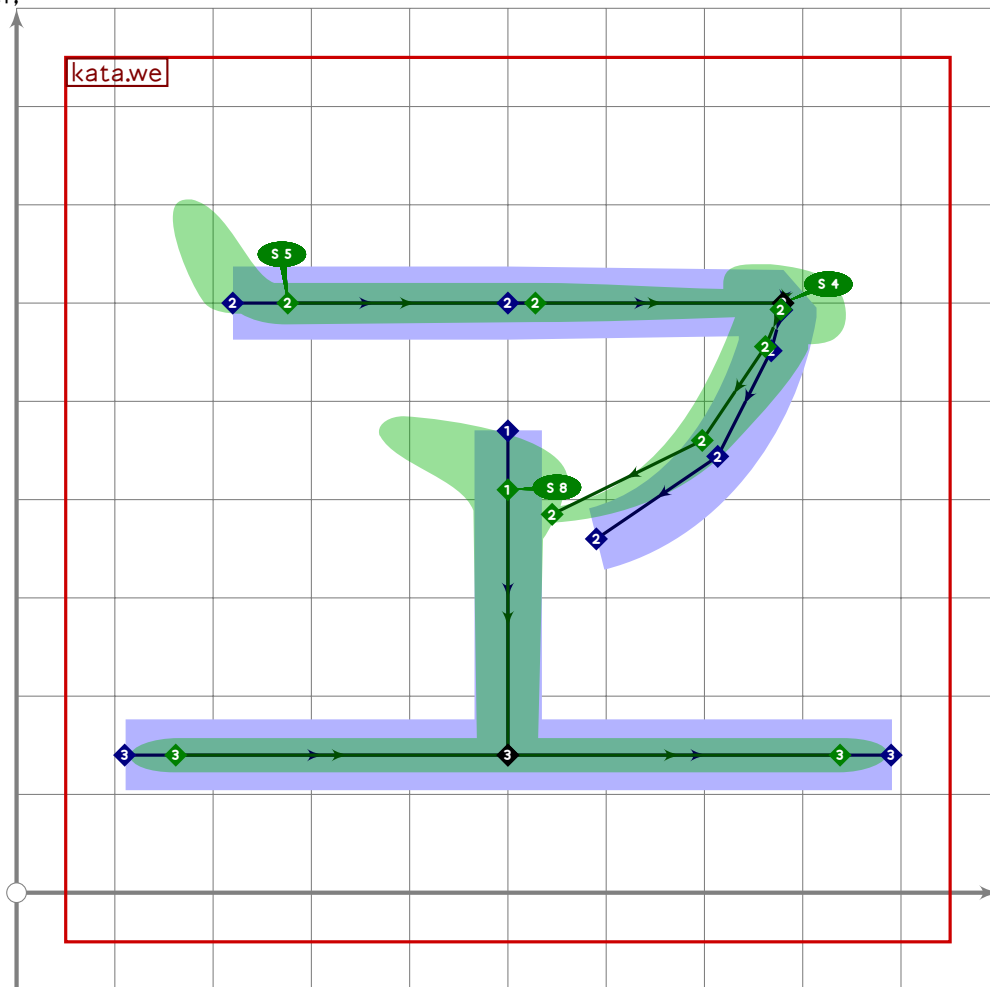
```

U+30F1
tsuku.uni30F1

```

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

```



KATA

```

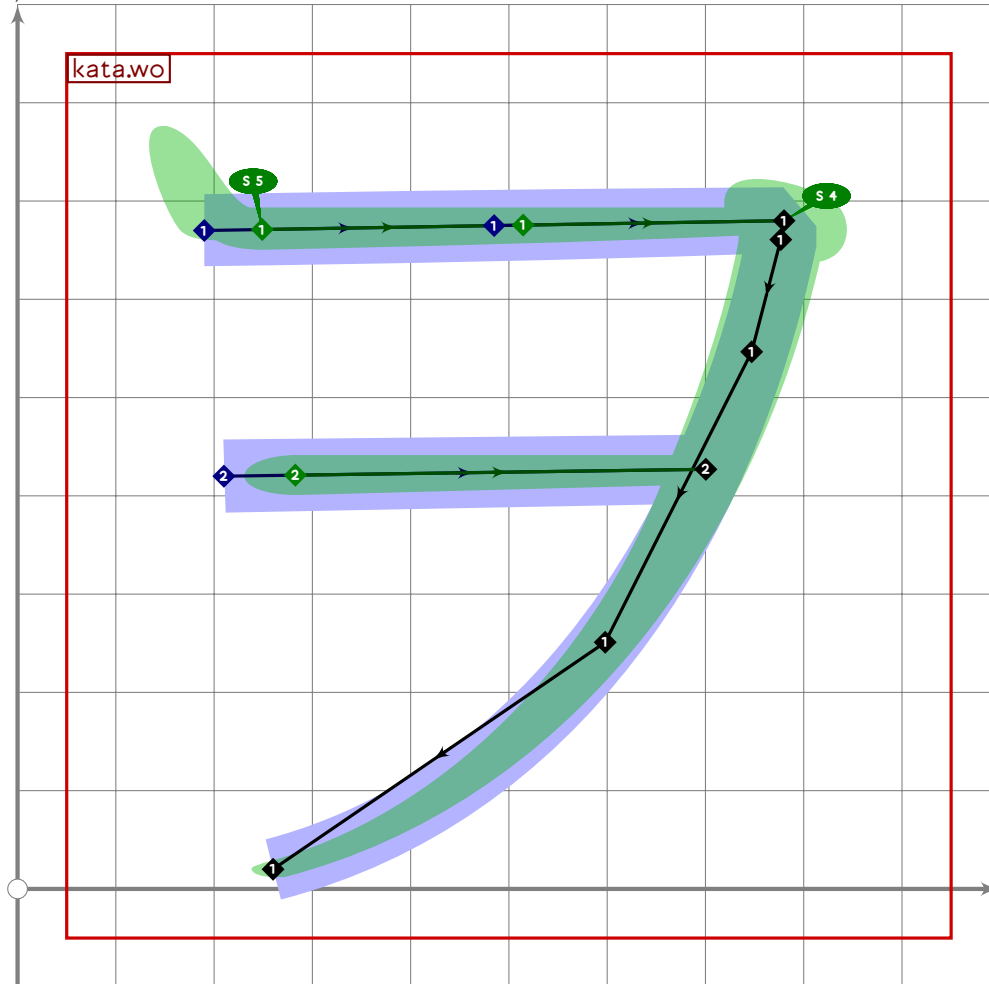
770
771 vardef kata.we =
772   push_pbox_toexpand("kata.we");
773
774   push_stroke((500,470-60*mincho)-(500,140),
775     (1.5,1.5)-(1.4,1.4));
776   set_boserif(0,0,8);
777
778   kata.fu_stroke((220,600),(780,600),

```

```

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

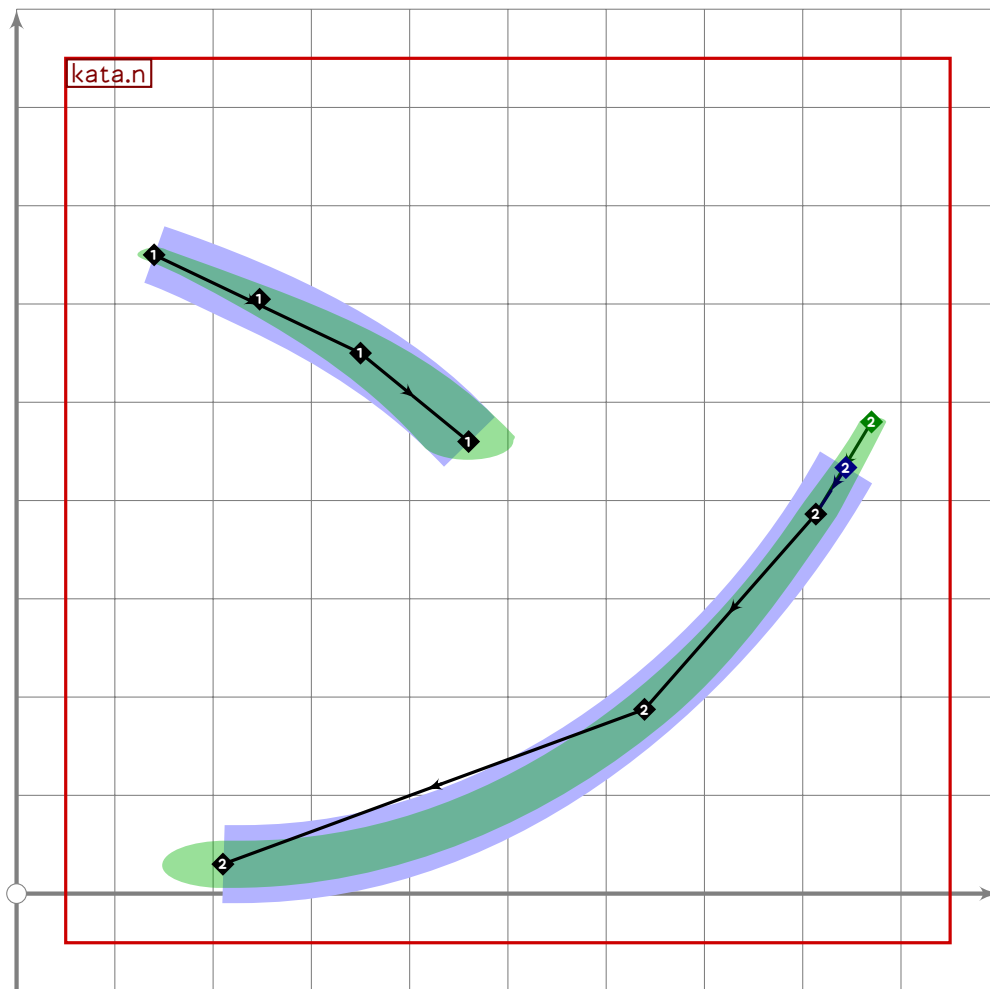
```



```

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

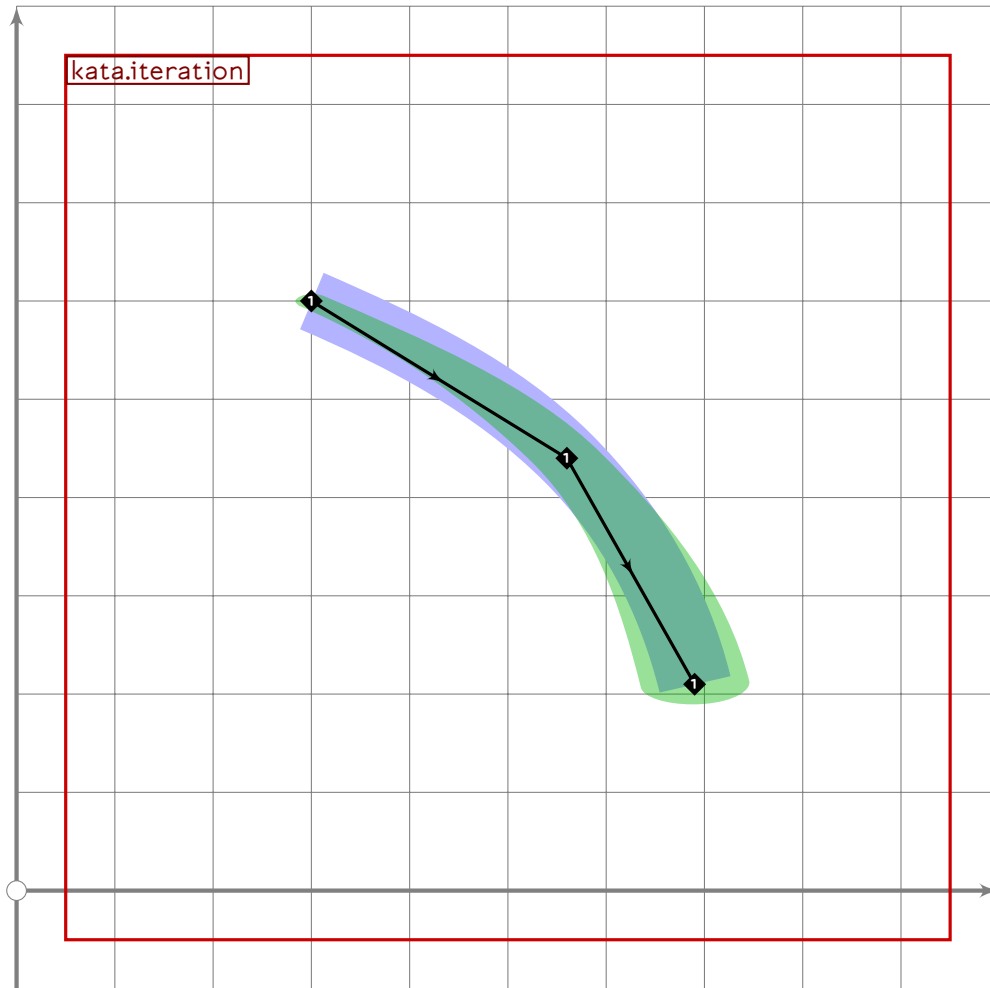
```



```

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

```

```

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

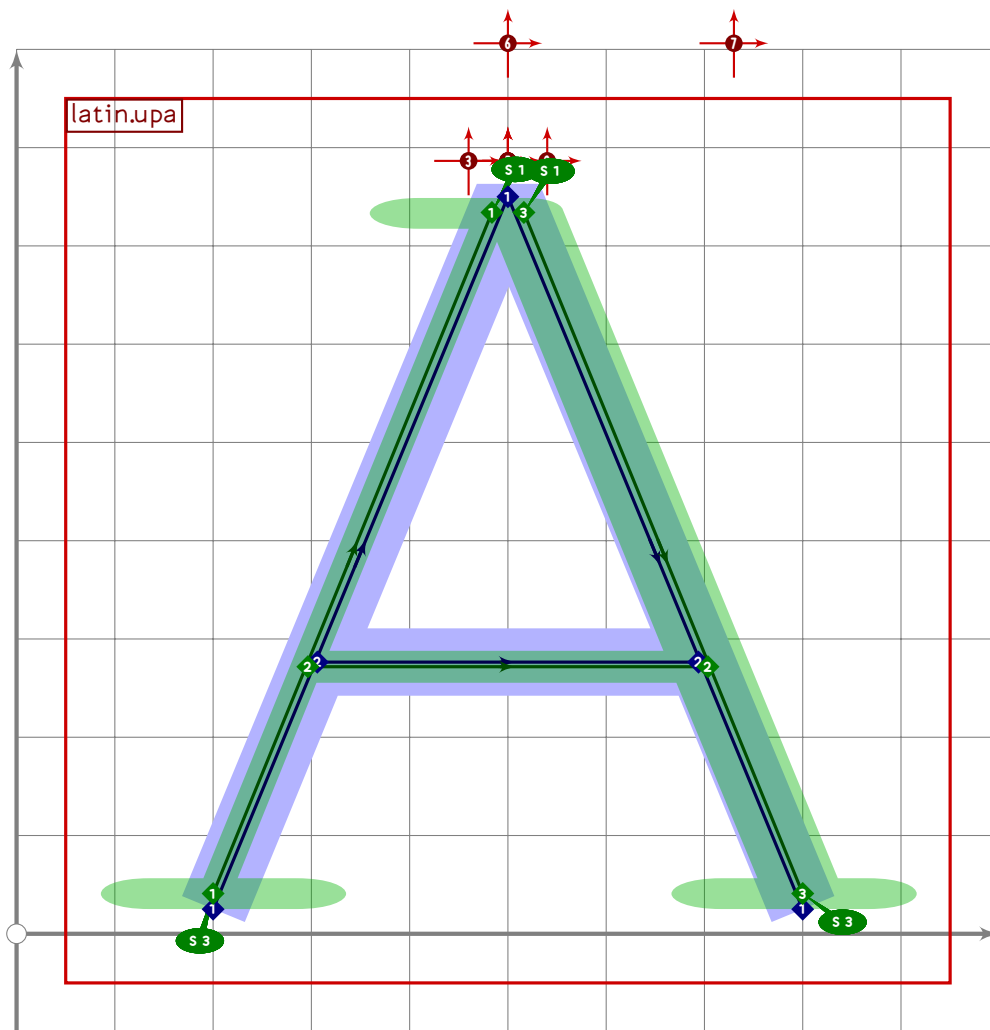
```

latin.mp

```

1 %
2 % Latin and related letters for Tsukurimashou
3 % Copyright (C) 2011, 2012, 2013, 2021 Matthew Skala
4 %
5-29 [Standard copyright notice]
30
31 inclusion_lock(latin);
32
33

```



LATI

```

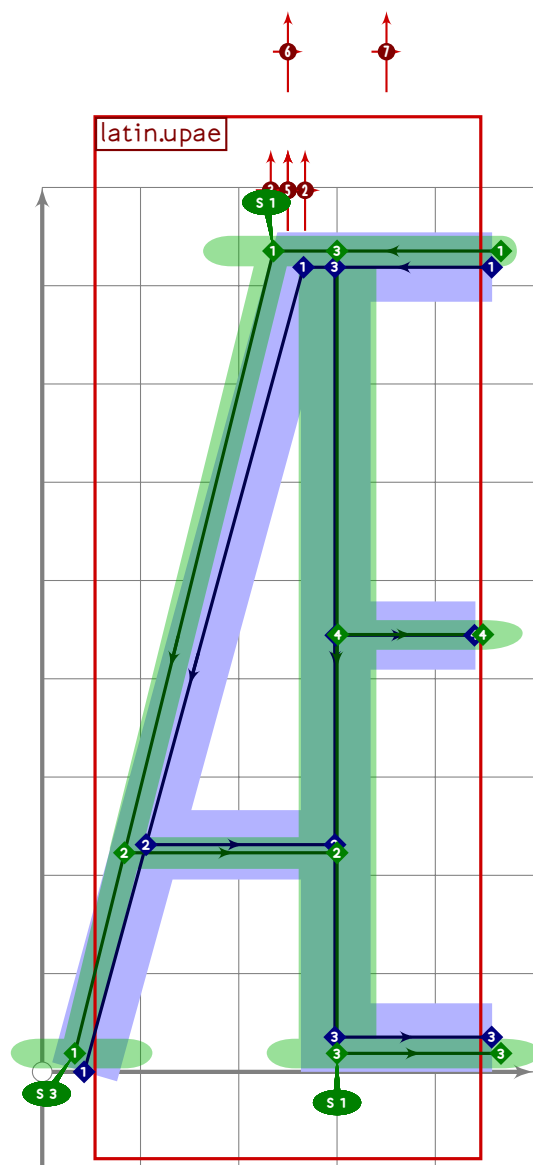
34
35 vardef latin.upa =
36   push_pbox_toexpand("latin.upa");
37   z1=(200,latin_wide_low_v);
38   z2=(500,latin_wide_high_v);
39   z3=(800,latin_wide_low_v);
40
41   if do_alteration:
42     z4=whatever[z1,(z2+alternate_adjust*left/2)]+(2,0);

```

```

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__bobrush(0,bralternate);
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__bobrush(0,bralternate);
54
55  push_stroke((z2+alternate_adjust*right/2)-z3,(1.6,1.6)-(1.6,1.6));
56  set__boserif(0,0,1);
57  set__boserif(0,1,3);
58  else:
59    z4=whatever[z1,z2]+(2,0);
60    z5=whatever[z2,z3]-(2,0);
61    y4=y5=vmetric(0.333);
62
63    push_stroke(z1-z2-z3,(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
64    set__botip(0,1,0);
65    set__boserif(0,0,3);
66    set__boserif(0,1,1);
67    set__boserif(0,2,3);
68
69    push_stroke(z4-z5,(1.6,1.6)-(1.6,1.6));
70  fi;
71
72  tsu__accent.shift__anchors(y part olda>vmetric(0.52))
73    (((0,0) transformed tsu__xf.cap_upper_accent)-
74    ((0,0) transformed accent_default[anc_upper]));
75  expand__pbox;
76 enddef;

```



```

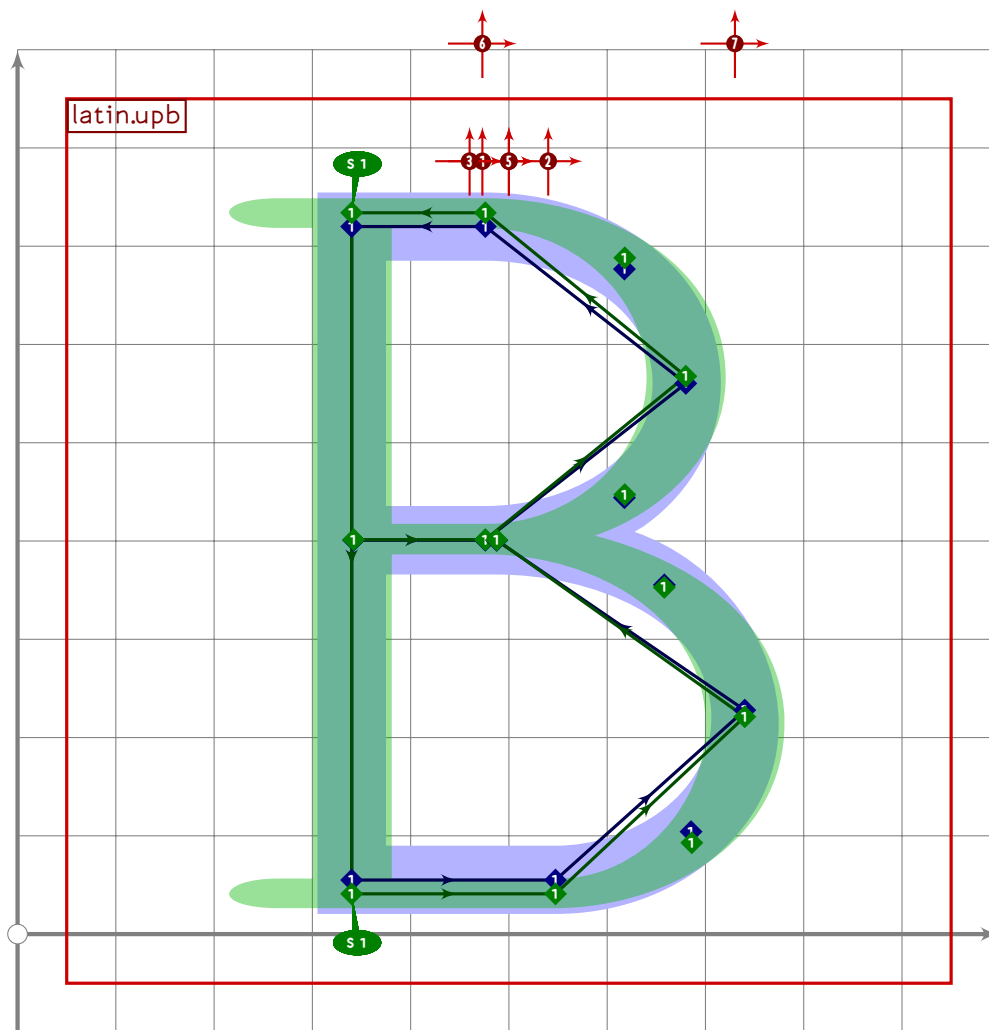
77
78 vardef latin.upae =
79   push_pbox_toexpand("latin.upae");
80   y1=y2=latin_wide_high_h;
81   y3=y4=latin_wide_low_h;
82   y5=y6=vmetric(0.522);
83
84   (x1+x7)/2=500;
85   x2=x3;
86   x1=x4;
87   x5=x2+2;
88   x6=0.89[x2,x1];
89   (x1-x2)=(y2-y3)*0.55;
90
91   y7=latin_wide_low_v;
92   x7=(-1.6)[x2,x1];

```

```

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

```



```

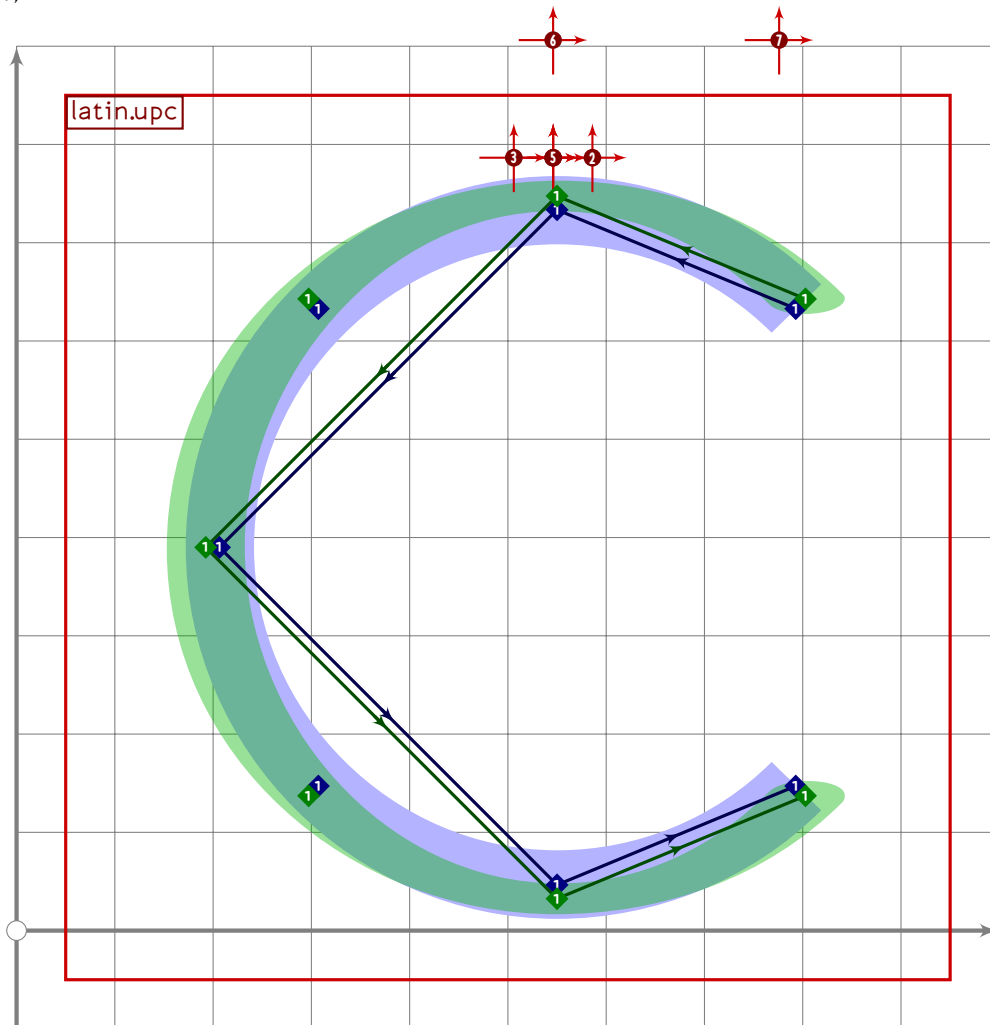
118
119 vardef latin.upb =
120   push_pbox_toexpand("latin.upb");
121   latin.upp_base(340);
122
123   x6=x5;
124   x7=0.61[x5,x3];
125   x8=x5+400;
126
127   y6=y7=latin_wide_low_h;
128   y8=0.5[y6,y1];
129
130   z9=z2;
131
132   replace_strokep(0)(oldp-z6-z7{right}..z8..{left}z9);
133   replace_strokep(0)(subpath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

```

```

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

```



```

147
148 vardef latin.upc =
149   push_pbox_toexpand("latin.upc");
150   push_stroke(
151     (subpath (0.5,3.5) of ((1,0)..(0,1)..(-1,0)..(0,-1)..cycle))
152     scaled ((latin_wide_high_r-latin_wide_low_r)/2)
153     shifted (centre_pt+(50,0)),
154     (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
155
156 tsu_accent.shift_anchors(ypart olda>vmetric(0.52))
157   (((0,0) transformed tsu_xf.cap_upper_accent)-

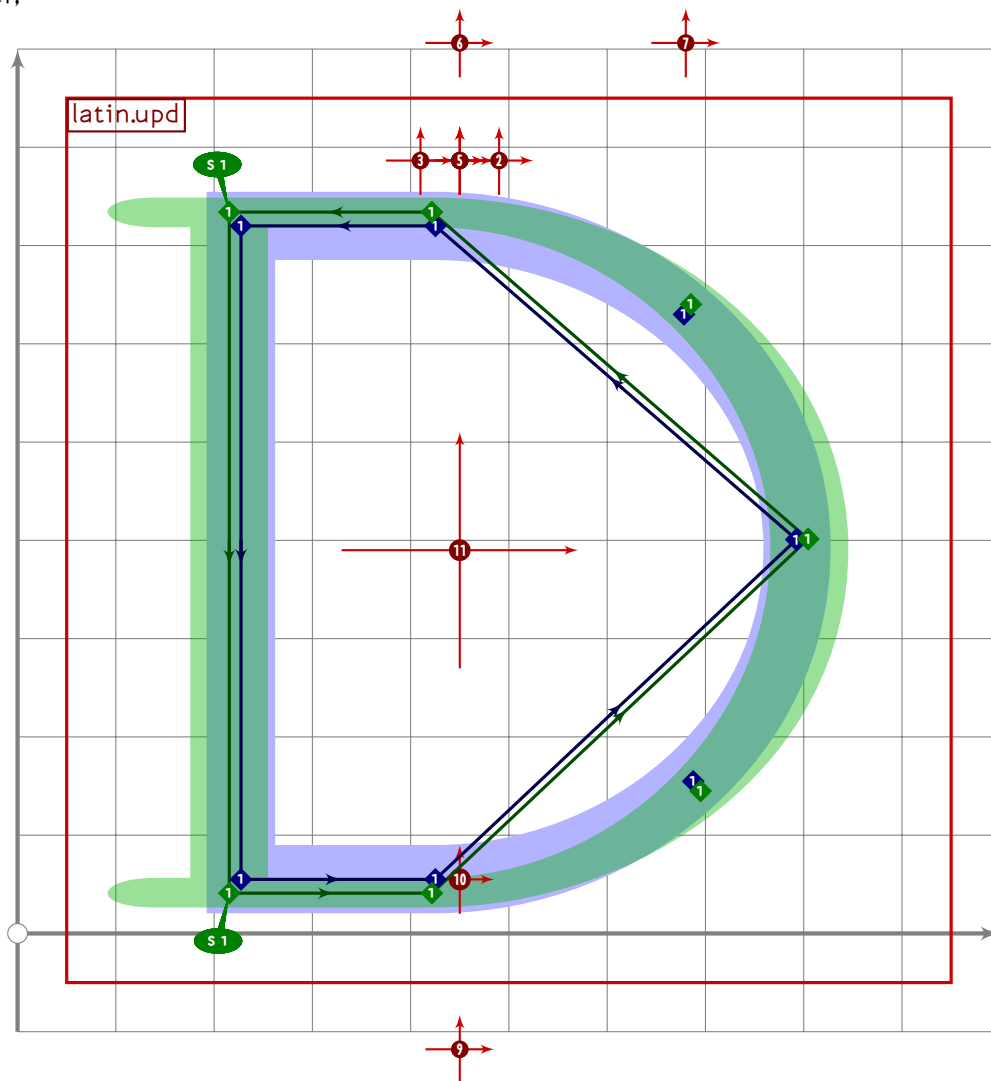
```

U+FF24
tsuku.uniFF24

```

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

```



```

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

```

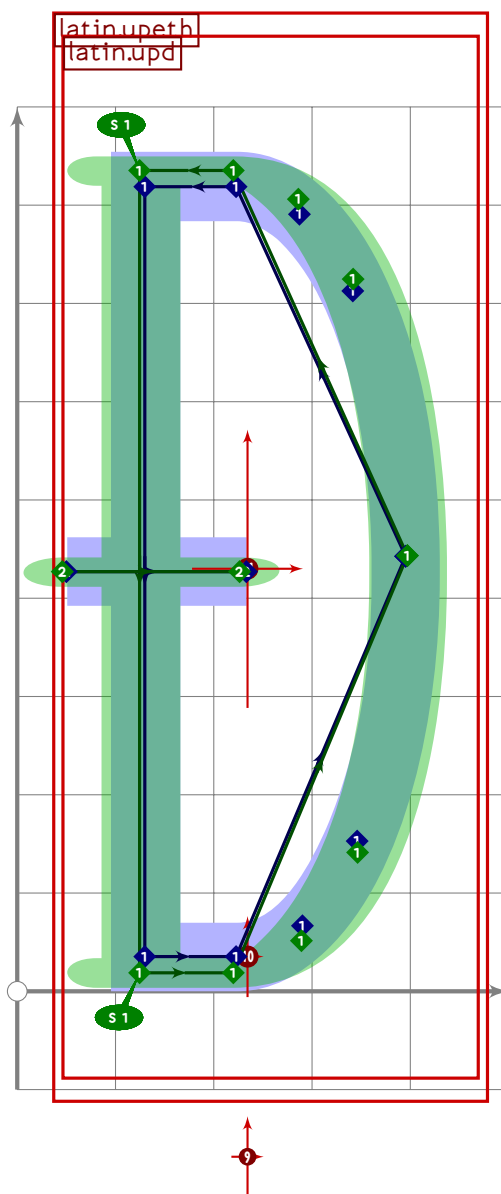
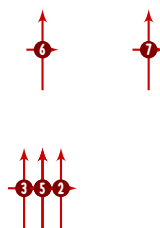
LATI


```

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

```

U+00D0
tsuku.Eth



LATI

```

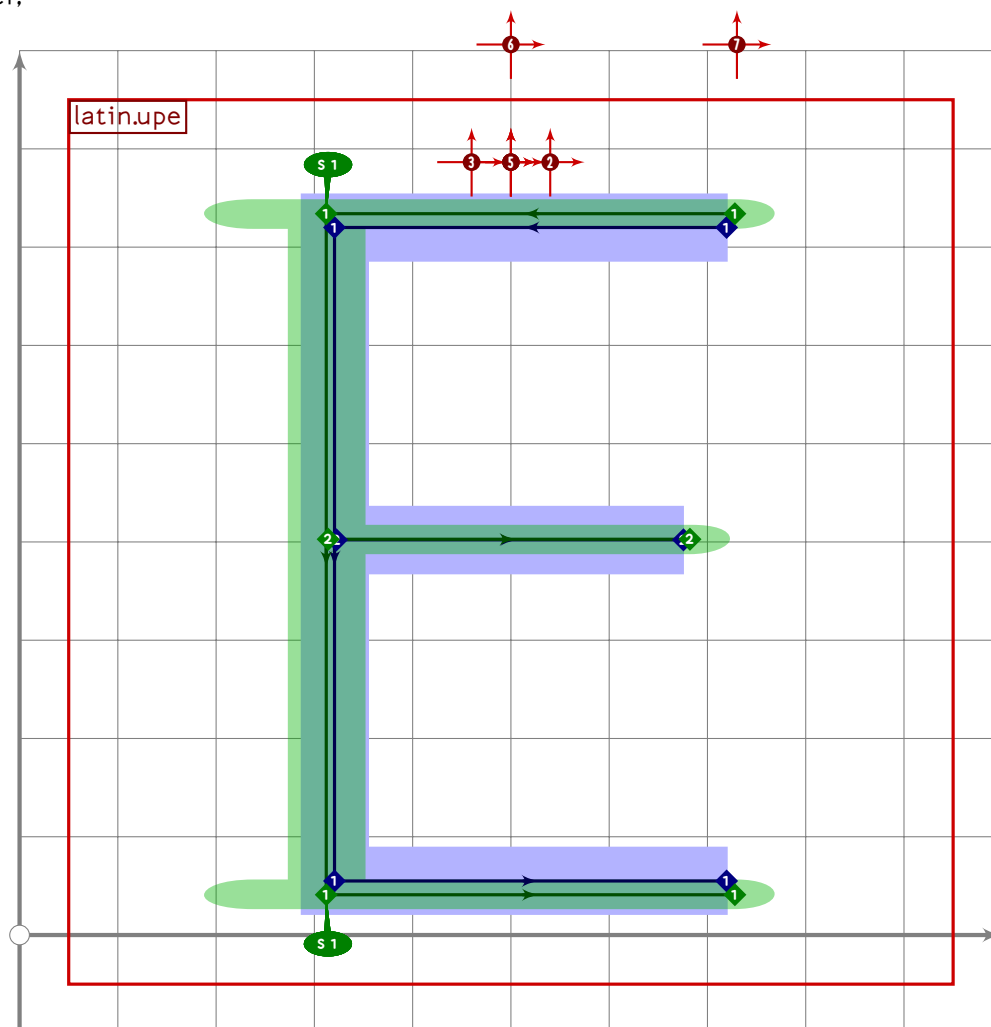
186
187 vardef latin.upeth =
188   push_pbox__toexpand("latin.upeth");
189   latin.upd;
190   push_stroke((0.5[z1,z2]+(-170,0))-((0.5[z1,z2]+(220,0)),
191     (1.6,1.6)-(1.6,1.6));

```

```

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

```



```

198
199 vardef latin.upe =
200   push_pbox_toexpand("latin.upe");
201   y1=y2=latin_wide_high_h;
202   y3=y4=latin_wide_low_h;
203   y5=y6=vmetric(0.522);
204
205   (x1+x2)/2=520;
206   x2=x3;
207   x1=x4;
208   x5=x2+2;
209   x6=0.89[x2,x1];
210   (x1-x2)=(y2-y3)*0.6;
211

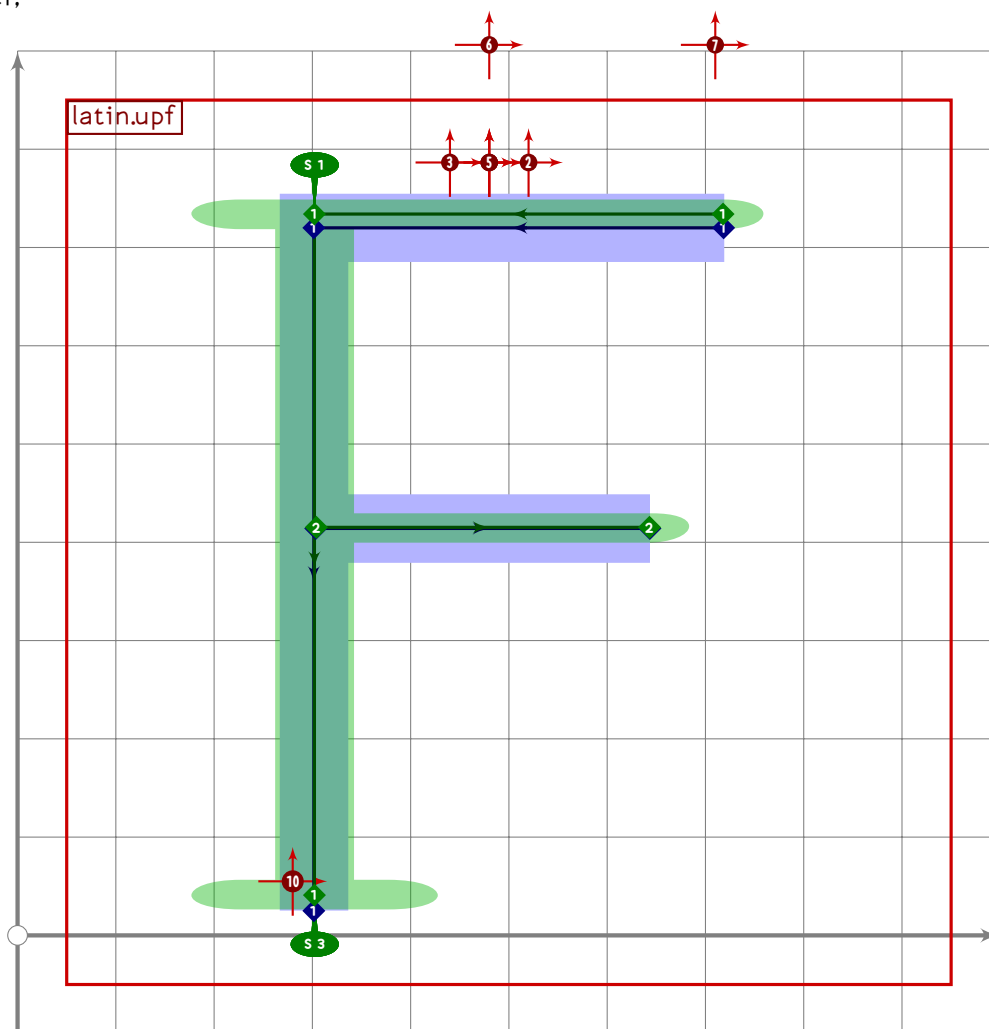
```

U+FF26
tsuku.uniFF26

```

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

```



LATI

```

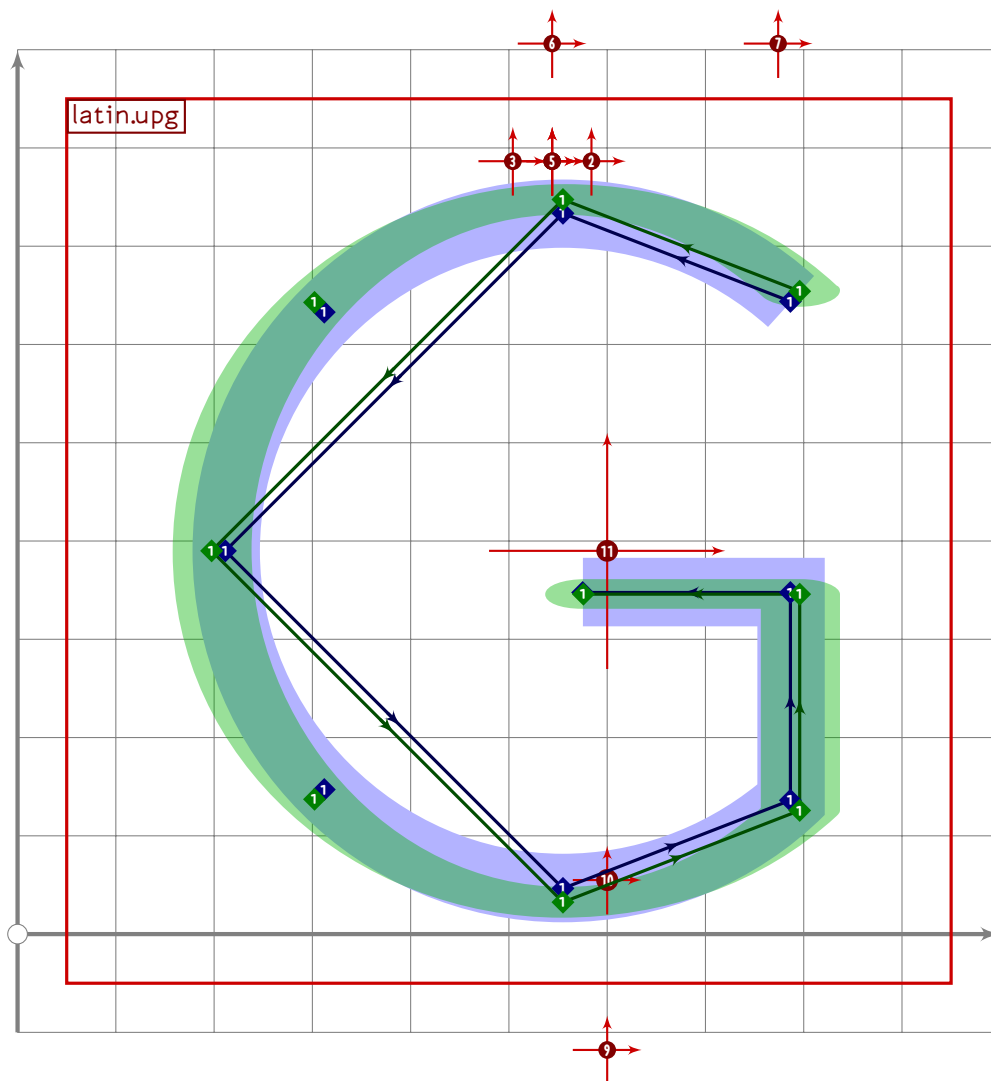
225
226 vardef latin.upf =
227   push_pbox_toexpand("latin.upf");
228   y1=y2=latin_wide_high_h;
229   y3=latin_wide_low_v;
230   y4=y5=vmetric(0.54);
231

```

```

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

```



```

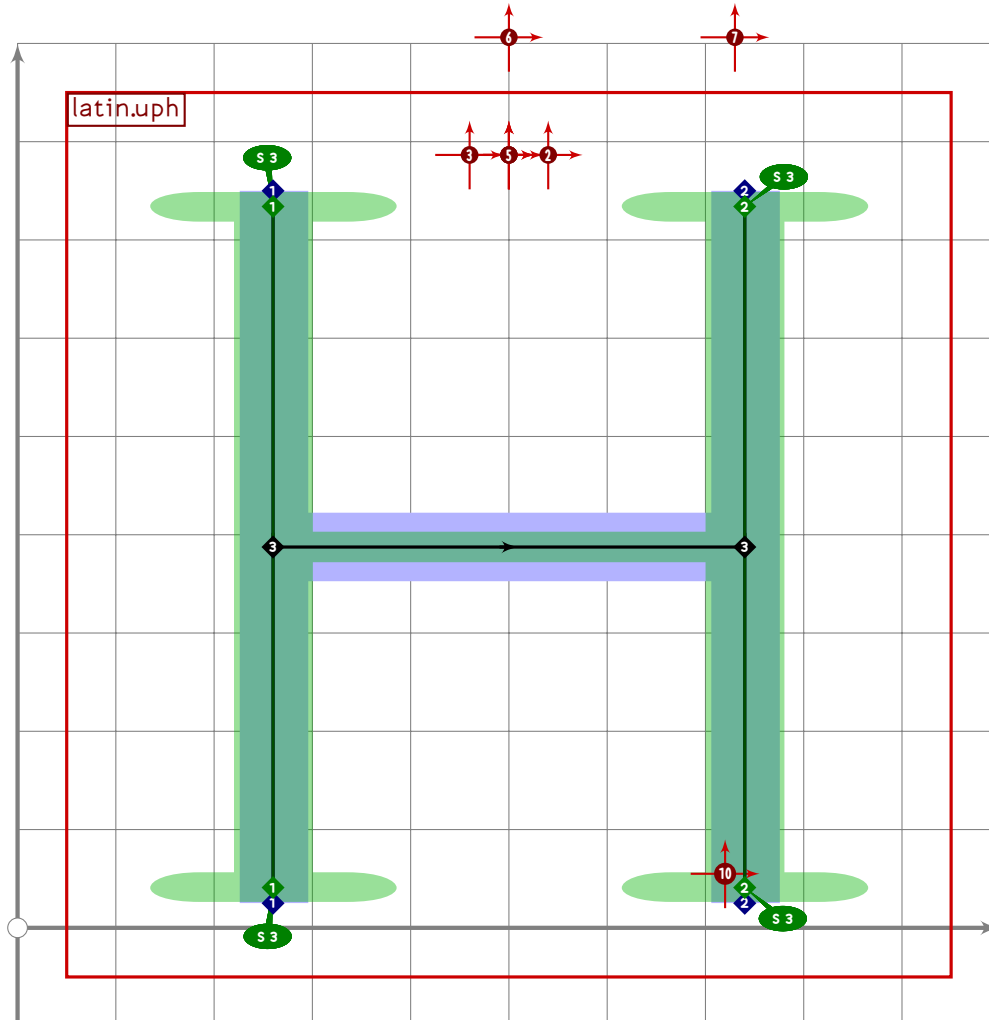
251
252 vardef latin.upg =
253   push_pbox_toexpand("latin.upg");
254   push_stroke(
255     (subpath (0.53,347) of ((1,0)..(0,1)..(-1,0)..(0,-1)..cycle))
256     scaled ((latin_wide_high_r-latin_wide_low_r)/2)
257     shifted (centre_pt+(55,0)),
258     (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-
259     (1.6,1.6)-(1.6,1.6));
260
261   x1=xpart point 4 of get_strokep(0);
262   y1=y2=vmetric(0.44);
263   x1-x2=y1-(ypart point 4 of get_strokep(0))*1.0;
264
265   replace_strokep(0)(oldp-z1-z2);
266   set_botip(0,4,1);
267   set_botip(0,5,1);
268-269
270   tsu_accent.shift_anchors(ypart olda>vmetric(0.52))

```

```

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

```



```

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

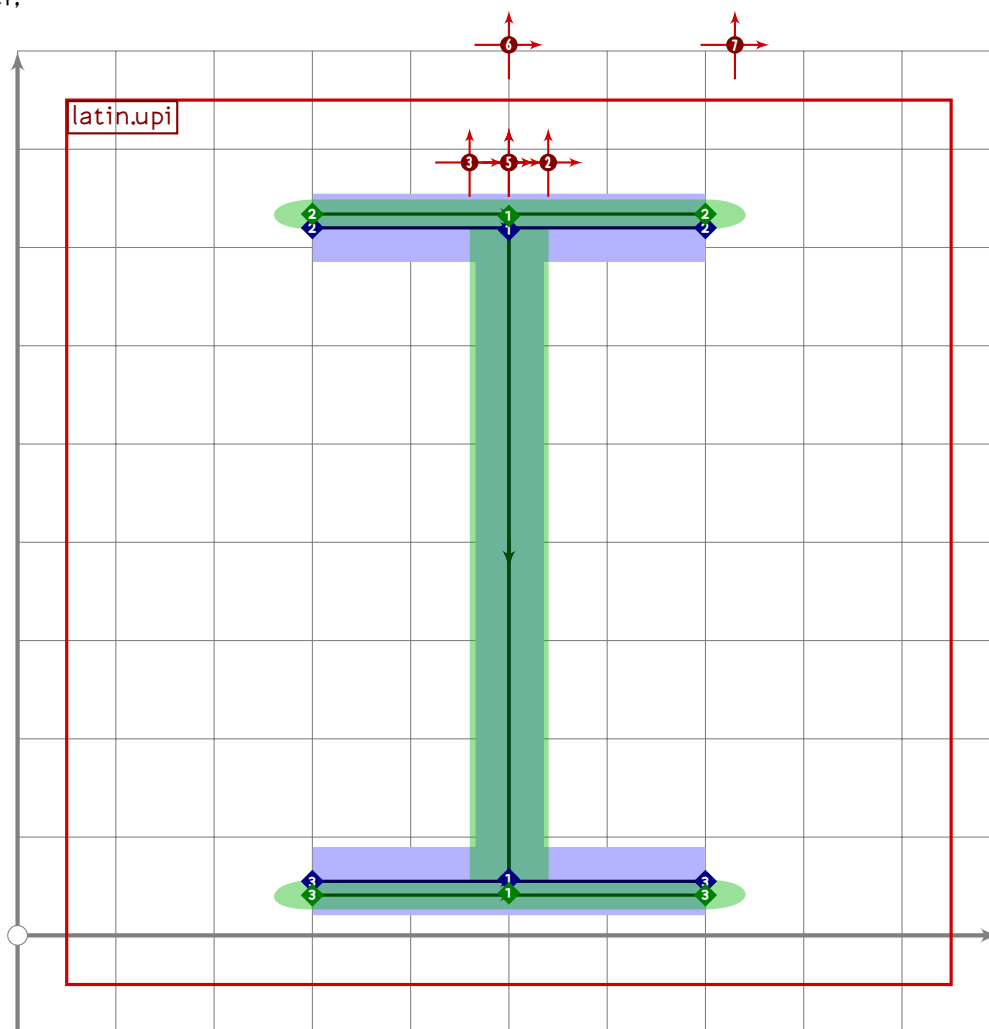
```

U+FF29
tsuku.uniFF29

```

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

```



LATI

```

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

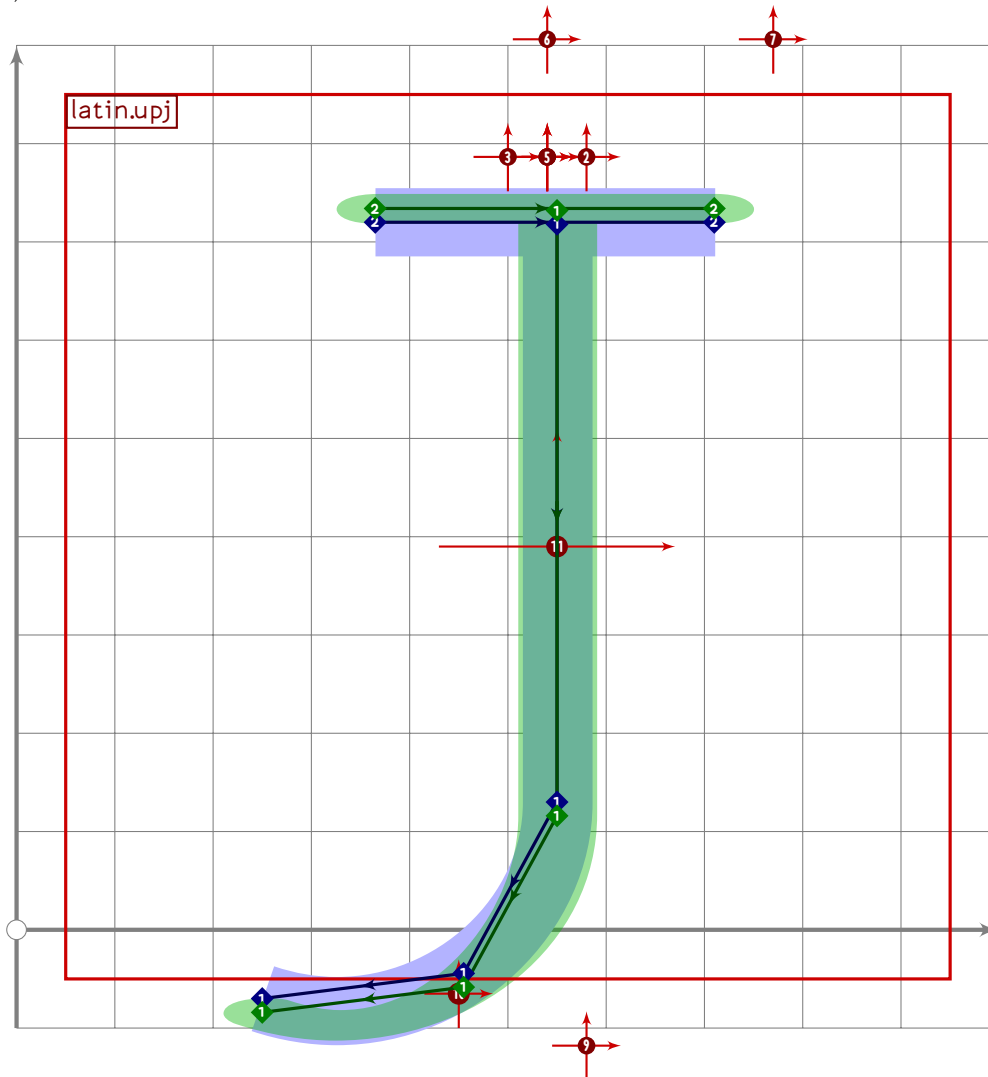
```



```

311 (1,6,1,6)-(1,6,1,6));
312
313 push_stroke((300,latin_wide_low_h)-(700,latin_wide_low_h),
314 (1,6,1,6)-(1,6,1,6));
315
316 tsu_accent.shift_anchors(ypart olda>vmetric(0.52))
317 (((0,0) transformed tsu_xf.cap_upper_accent)-
318 ((0,0) transformed accent_default[anc_upper]));
319 expand_pbox;
320 enddef;

```



```

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

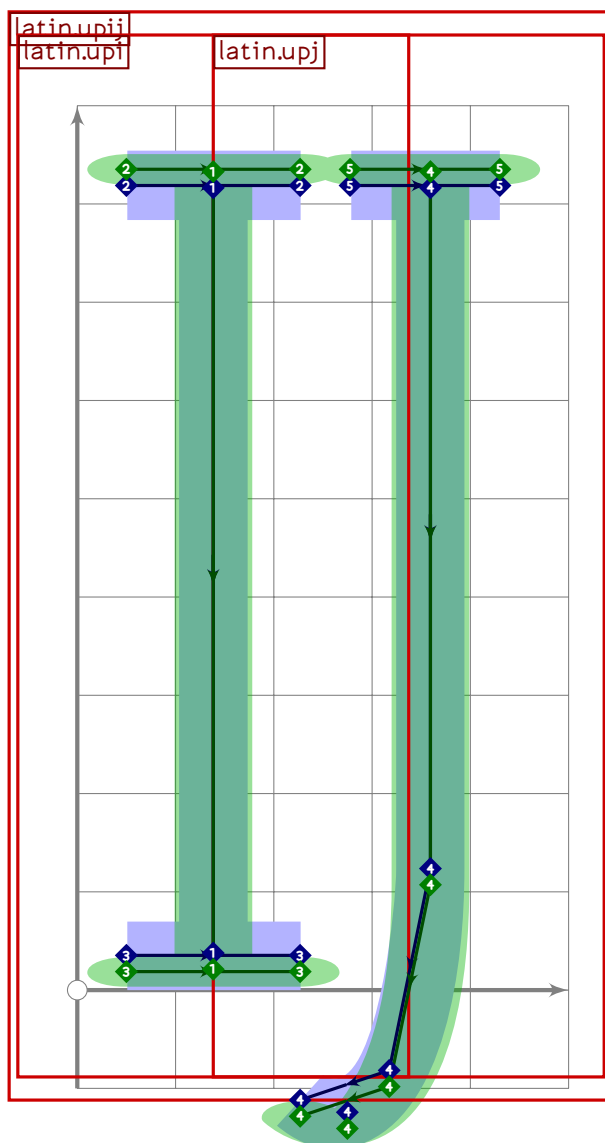
```

U+0132
tsuku.IJ

```

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

```



LATI

```

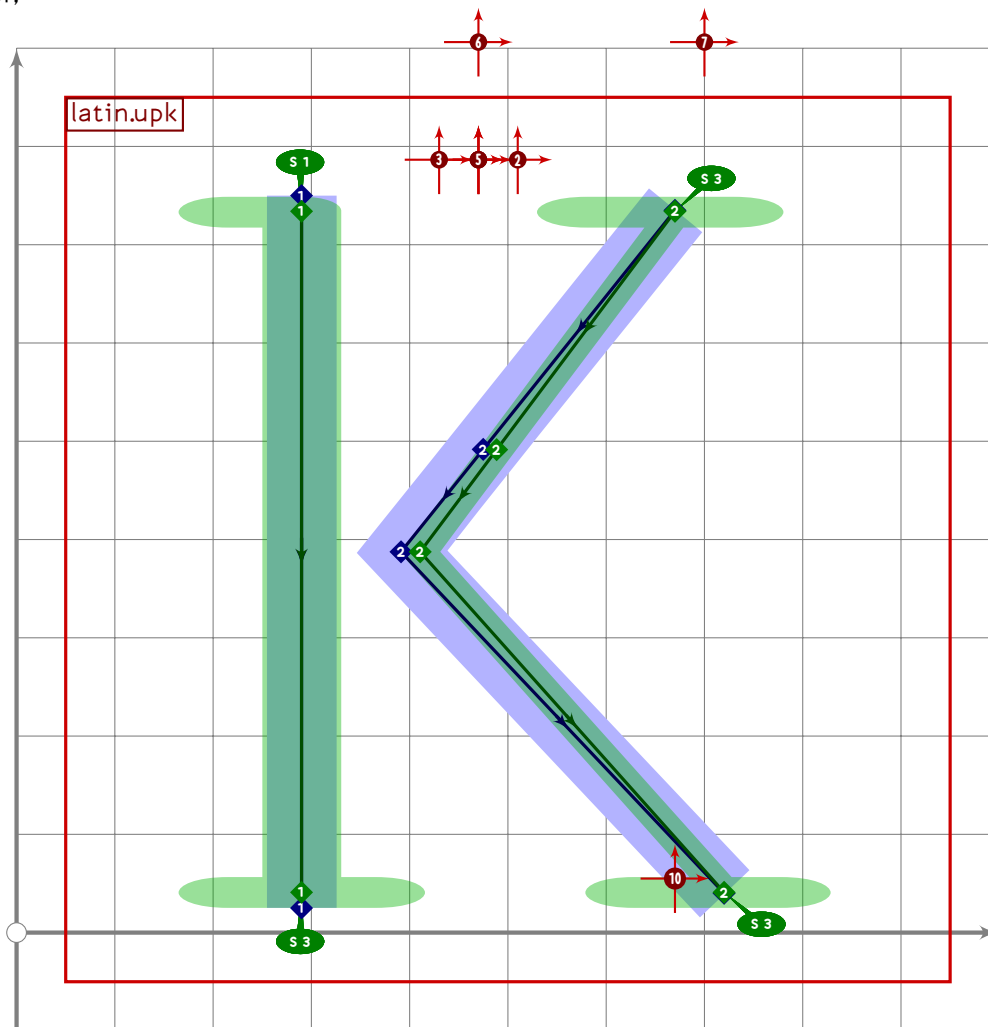
344
345 vardef latin.upij =

```

```

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

```



```

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

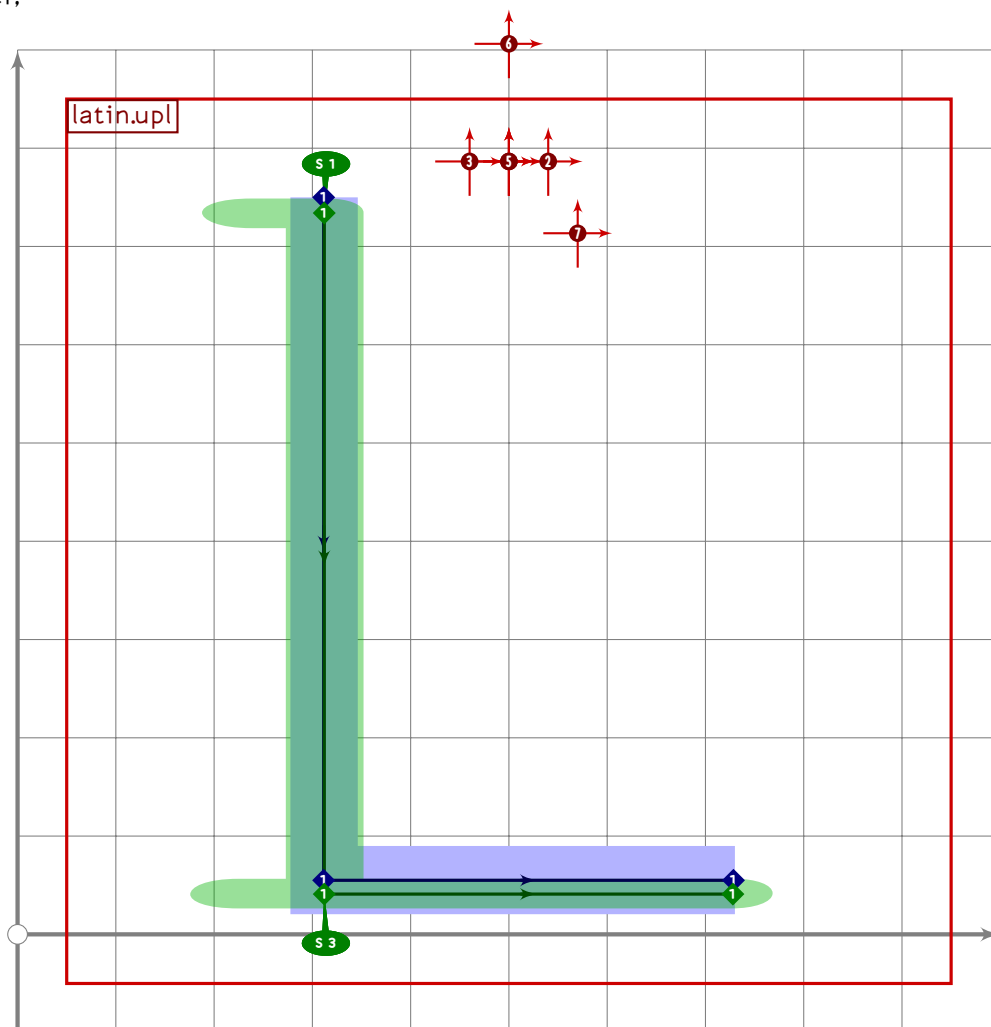
```

U+FF2C
tsuku.uniFF2C

```

366 set_boserif(0,0,1);
367 set_boserif(0,1,3);
368
369 push_stroke(z3-(0.7[z3,z4])-z4-z5,
370   (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
371 set_botip(0,2,1);
372 if do_alteration:
373   set_bobrush(0,bralterate);
374   set_boserif(0,0,3);
375   set_boserif(0,3,3);
376 fi;
377
378 tsu_accent.shift_anchors(ypart olda>vmetric(0.52))
379   (((0,0) transformed tsu_xf.cap_upper_accent)-
380    ((0,0) transformed accent_default[anc_upper])+(-30,0));
381 tsu_accent.shift_anchors(ai=anc_lower_connect)((170,0));
382 expand_pbox;
383 enddef;

```



LATI

```

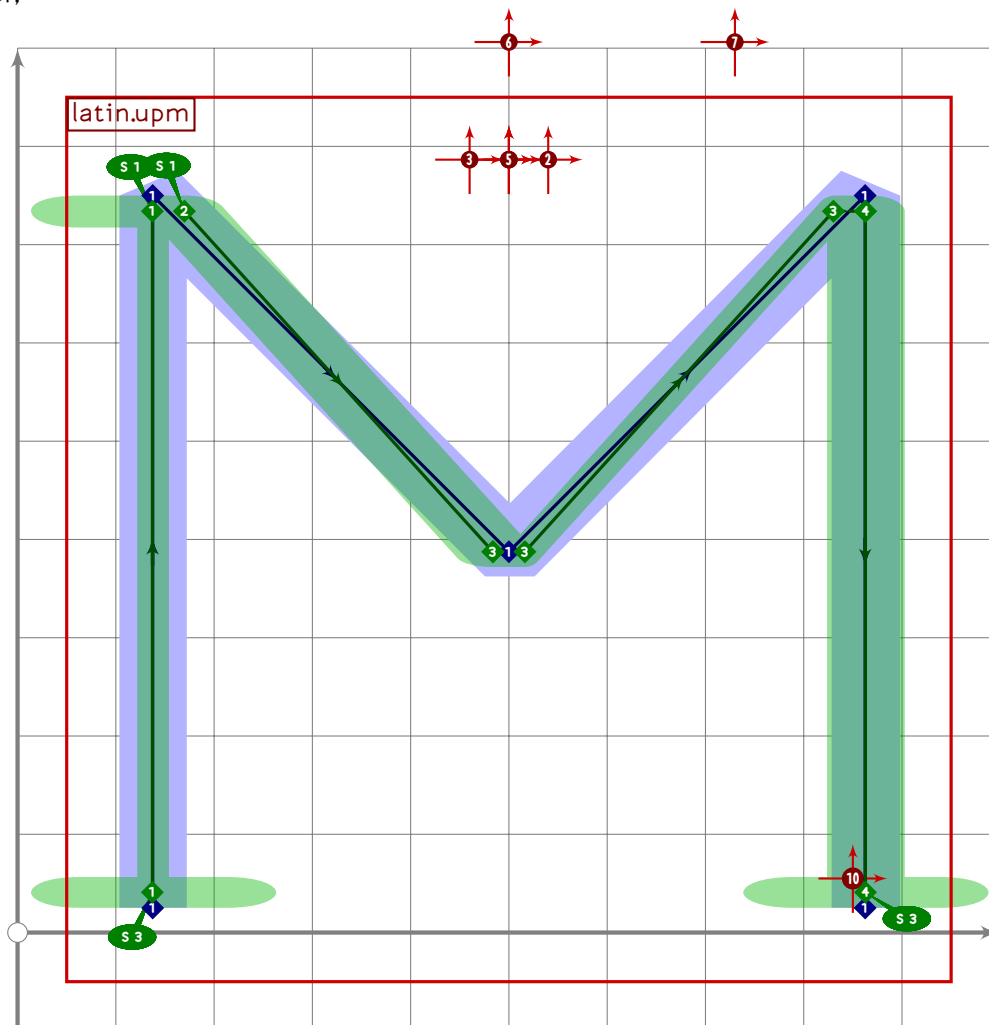
384
385 vardef latin.upl =

```

```

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

```



LATI

405

```

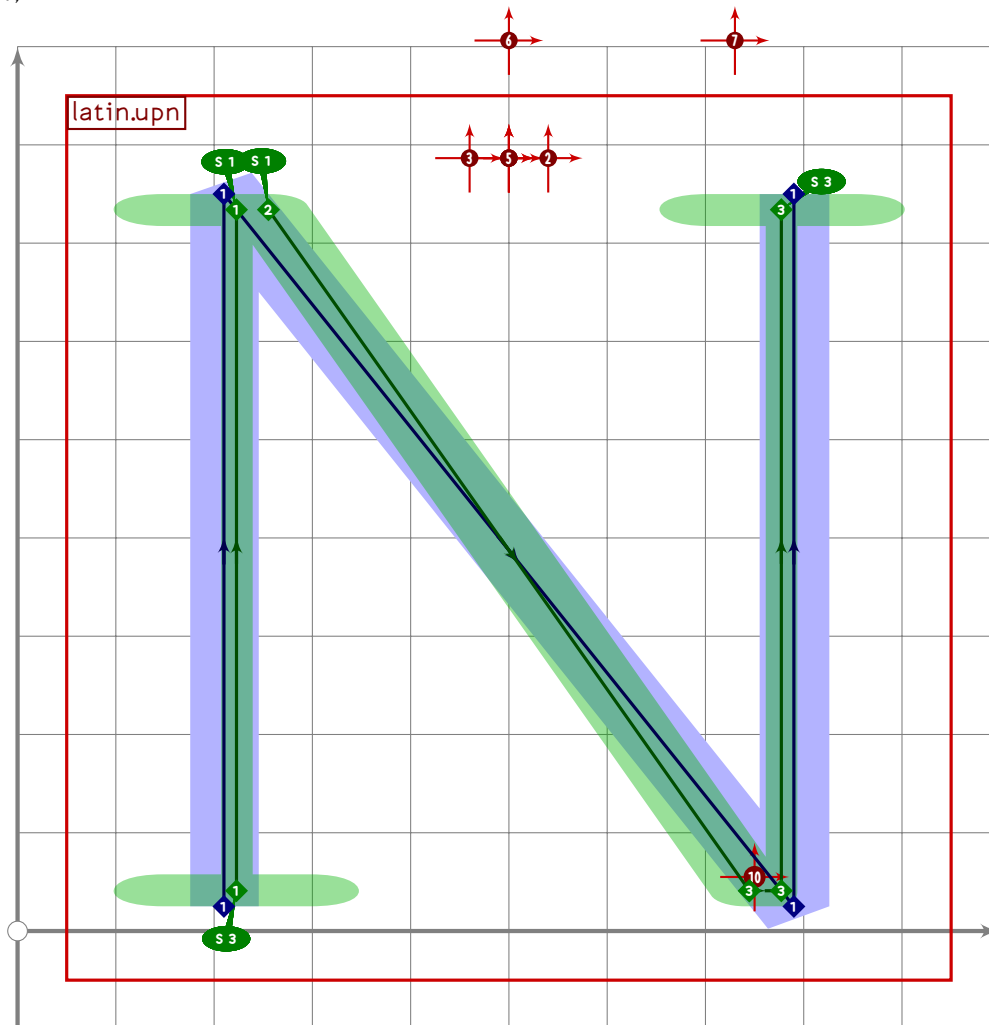
406 vardef latin.upm =
407   push_pbox_toexpand("latin.upm");
408   y1=y5=latin_wide_low_v;
409   y2=y4=latin_wide_high_v;
410   y3=(y1+y2)/2;
411
412   if do_alteration:
413     x1=x2;
414     x3=500+alternate_adjust/2;
415     x4=x5;
416     (x3-x1)=(x5-x3);
417
418     (x5-x1)=(y2-y1);
419
420   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

```

```

454 tsu_accent.shift_anchors(ypart olda>vmetric(0.52))
455 (((0,0) transformed tsu_xf.cap_upper_accent)-
456 ((0,0) transformed accent_default[anc_upper]));
457 tsu_accent.shift_anchors(ai=anc_lower_connect)((350,0));
458 expand_pbox;
459 enddef;

```



```

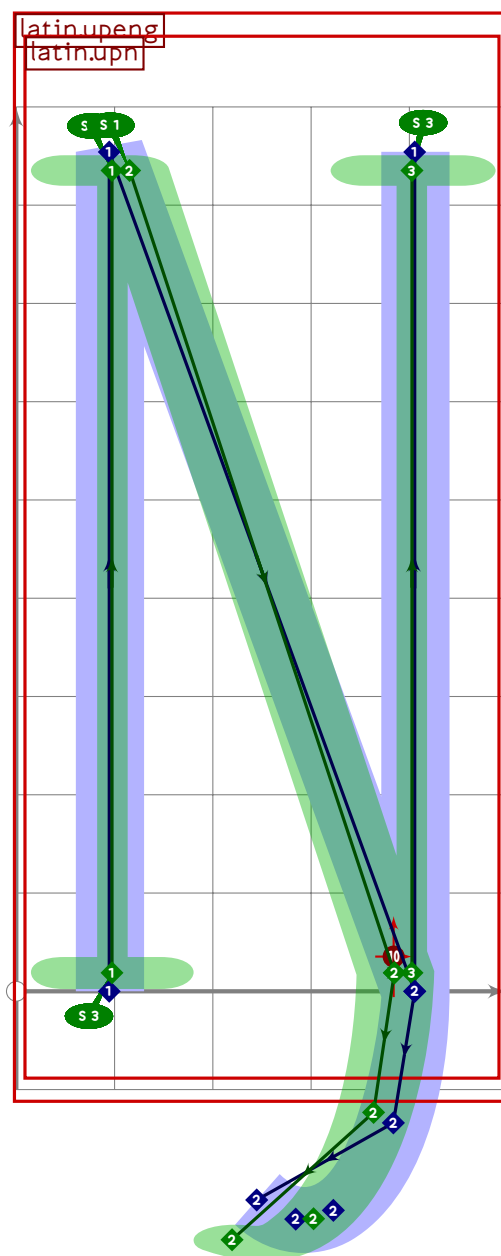
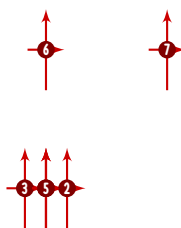
460
461 vardef latin.upn =
462   push_pbox_toexpand("latin.upn");
463   y1=y3=latin_wide_low_v;
464   y2=y4=latin_wide_high_v;
465
466   x1=x2;
467   x3=x4;
468   (x1+x3)/2=500;
469   (x3-x1)=(y2-y1)*4/5;
470
471   if do_alteration:
472     push_stroke(z1-z2,(1.6,1.6)-(1.6,1.6));
473     set_boserif(0,0,3);

```

```

474     set__boserif(0,1,1);
475     set__bobrush(0,bralternate);
476
477     push__stroke((z2+alternate_adjust*right)-(z3+alternate_adjust*left),
478         (1.6,1.6)-(1.6,1.6));
479     set__boserif(0,0,1);
480
481     push__stroke((z3+alternate_adjust*left)-z3-z4,
482         (1.6,1.6)-(1.6,1.6)-(1.6,1.6));
483     set__boserif(0,2,3);
484     set__bobrush(0,bralternate);
485     else:
486         push__stroke(z1-z2-z3-z4,(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
487         set__botip(0,1,0);
488         set__botip(0,2,0);
489         set__boserif(0,0,3);
490         set__boserif(0,1,1);
491         set__boserif(0,3,3);
492     fi;
493
494     tsu_accent.shift_anchors(ypart olda>vmetric(0.52))
495         (((0,0) transformed tsu_xf.cap_upper_accent)-
496         ((0,0) transformed accent_default[anc_upper]));
497     tsu_accent.shift_anchors(ai=anc_lower_connect)((250,0));
498     expand__pbox;
499 enddef;

```

LATI

```

500
501 vardef latin.upeng =
502   push_pbox_toexpand("latin.upeng");
503   latin.upn;
504   y5=latin_wide_desc_h;

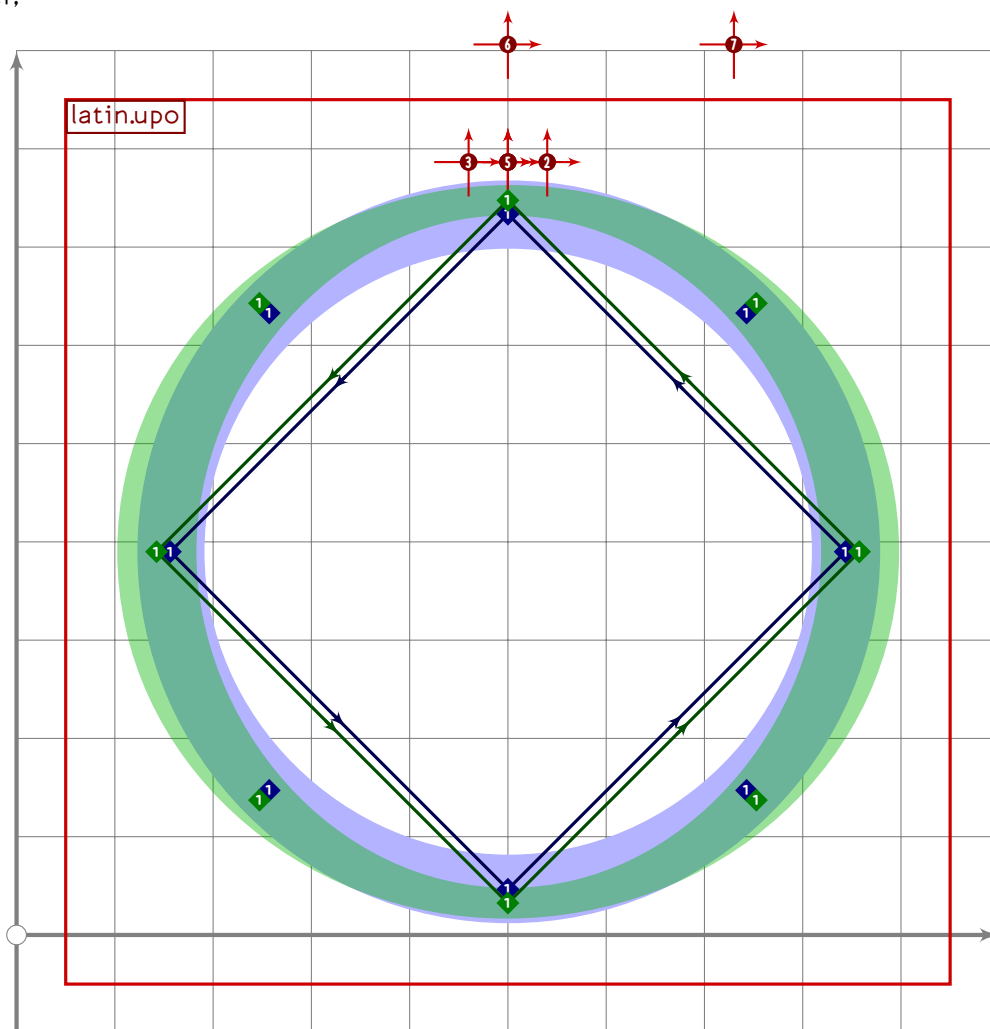
```

U+FF2F
tsuku.uniFF2F

```

505 if do_alteration:
506     x5=x3-alternate_adjust-300;
507     replace_strokep(-1)(oldp{dir 268}{.curl 0.8}z5);
508     replace_strokep(-1)(insert_nodes(oldp)(1.3));
509     replace_strokeq(-1)(oldq-(1.6,1.6)-(1.6,1.6));
510 else:
511     x5=x3-300;
512     push_stroke(z3{dir 268}{.curl 0.8}z5,(1.6,1.6)-(1.6,1.6));
513     replace_strokep(0)(insert_nodes(oldp)(0.3));
514 fi;
515
516 tsu_accent.shift_anchors(ypart olda>vmetric(0.52))
517     (((0,0) transformed tsu_xf.cap_upper_accent)-
518     ((0,0) transformed accent_default[anc_upper]));
519 expand_pbox;
520 enddef;

```



```

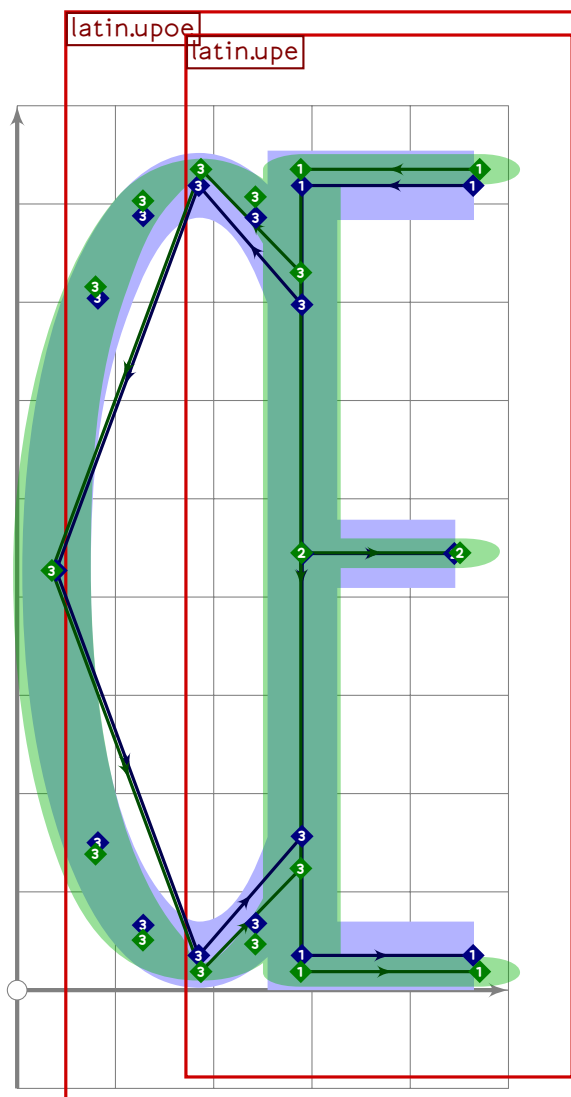
521
522 vardef latin.upo =
523     push_pbox_toexpand("latin.upo");
524     push_stroke(

```

```

525 ((1,0)..(0,1)..(-1,0)..(0,-1)..cycle)
526 scaled ((latin_wide_high_r-latin_wide_low_r)/2)
527 shifted centre_pt,
528 (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-cycle);
529
530 tsu_accent.shift_anchors(ypart olda>vmetric(0.52))
531 (((0,0) transformed tsu_xf.cap_upper_accent)-
532 ((0,0) transformed accent_default[anc_upper]));
533 expand_pbox;
534 endif;

```



```

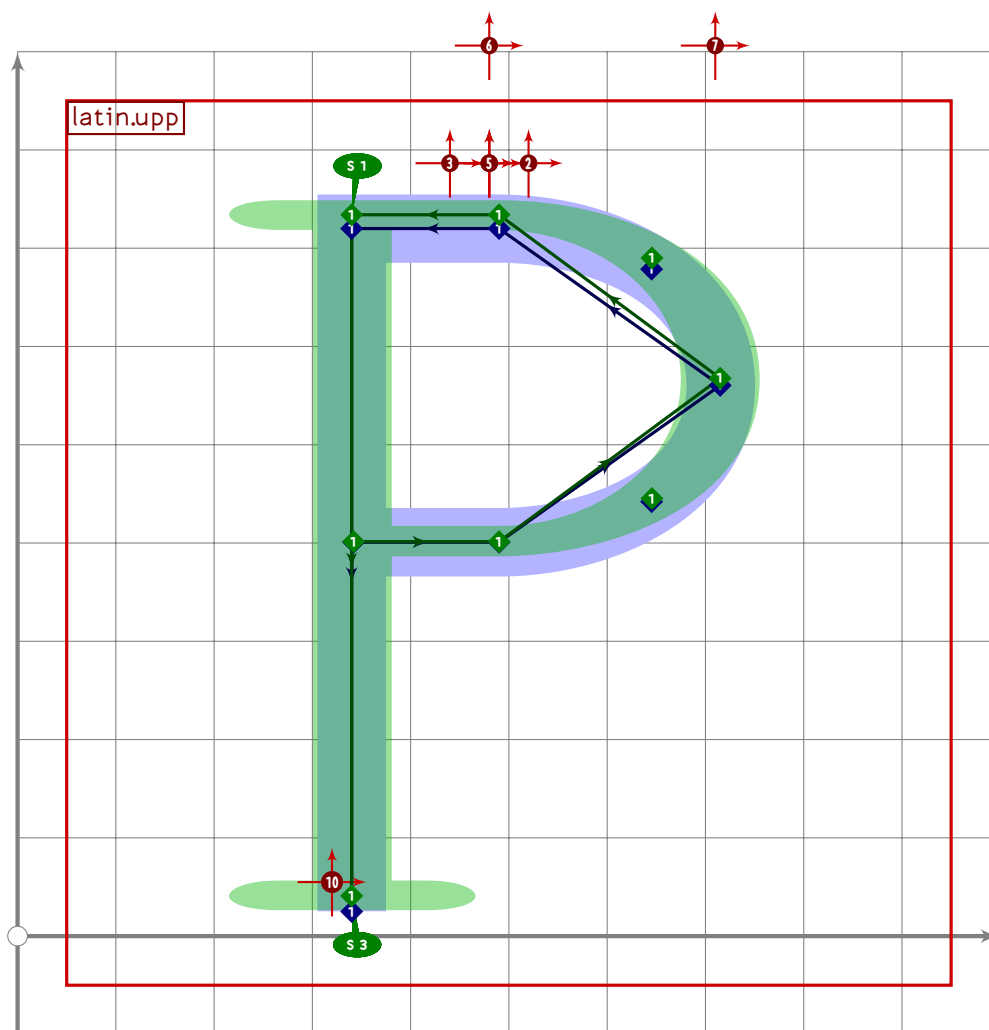
535
536 vardef latin.upoe =
537   push_pbox_toexpand("latin.upoe");
538   tsu_xform(identity shifted (280,0))(latin.upe);
539   set_boserif(-1,1,whatever);
540   set_boserif(-1,2,whatever);
541   push_stroke(
542     ((1,0)..(0,1)..(-1,0)..(0,-1)..(1,0))

```

```

543     scaled ((latin_wide_high_h-latin_wide_low_h)/2)
544     shifted (360,0.5[latin_wide_high_h,latin_wide_low_h]),
545     (1.2,1.2)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.2,1.2));
546 replace_strokep(0)(
547     subpath (xpart (oldp intersectiontimes get_strokep(-2)),
548         4-xpart ((reverse oldp) intersectiontimes get_strokep(-2)))
549     of oldp);
550
551 tsu_accent.shift_anchors(ypart olda>vmetric(0.52))
552     (((0,0) transformed tsu_xf.cap_upper_accent)-
553     ((0,0) transformed accent_default[anc_upper]));
554 expand_pbox;
555 enddef;
556
557 vardef latin_upp_base(expr b) =
558     x1=x5+2;
559     x5=340;
560     x2=x4=x5+b*0.4;
561     x3=x5+b;
562
563     y1=y2=vmetric(0.52);
564     y3=(y2+y4)/2;
565     y4=y5=latin_wide_high_h;
566
567     push_stroke(z1-z2{right}..z3.{left}z4-z5,
568         (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
569 enddef;

```

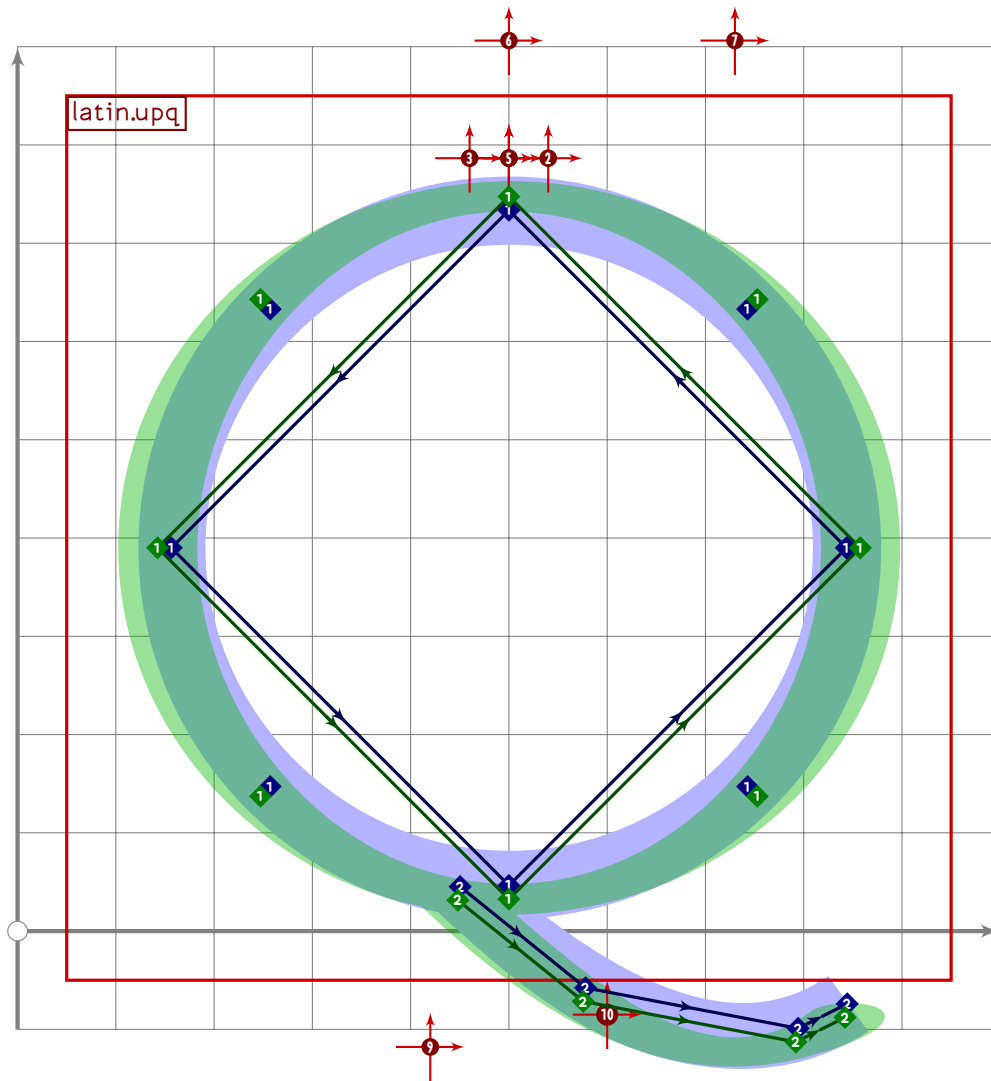


```

570
571 vardef latin.upp =
572   push_pbox_toexpand("latin.upp");
573   latin.upp_base(375);
574   replace_strokep(0)(oldp-(xpart point infinity of oldp,latin_wide_low_v));
575   replace_strokeq(0)(oldq-(1.6,1.6));
576   set_botip(0,4,1);
577   set_boserif(0,4,1);
578   set_boserif(0,5,3);
579
580   tsu_accent.shift_anchors(ypart olda>vmetric(0.52))
581     (((0,0) transformed tsu_xf.cap_upper_accent)-
582      ((0,0) transformed accent_default[anc_upper])+(-20,0));
583   tsu_accent.shift_anchors(ai=anc_lower_connect)((-180,0));
584   expand_pbox;
585 enddef;

```

LATI



```

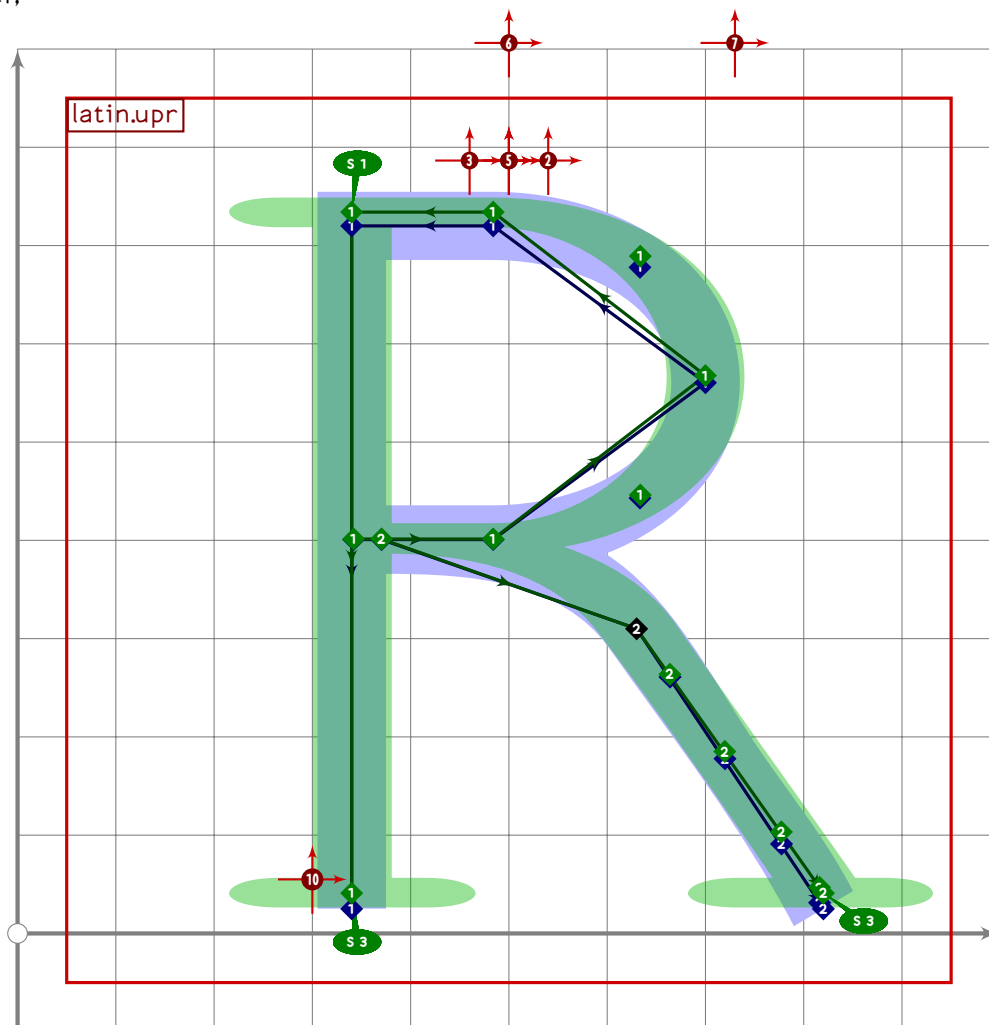
586
587 vardef latin.upq =
588   push_pbox_toexpand("latin.upq");
589   push_stroke(((1,0)..(0,1)..(-1,0)..(0,-1)..cycle)
590     scaled ((latin_wide_high_r-latin_wide_low_r)/2)
591     shifted centre_pt,
592     (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-cycle);
593
594   z1=point 2.9 of get_stroke(0);
595   z2=z1+(350,150);
596   z3=z2+(50,25);
597   push_stroke(subpath (0,0,2,2) of (z1{curl 0}..z2..z3),
598     (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
599   replace_stroke(0)(insert_nodes(olddp)(0.4));
600
601   tsu_accent.shift_anchors(ypart olda>vmetric(0.52))
602     (((0,0) transformed tsu_xf.cap_upper_accent)-
603     ((0,0) transformed accent_default[anc_upper]));
604   tsu_accent.shift_anchors(ai=anc_lower)((-80,0));

```

```

605 tsu_accent.shift_anchors(ai=anc_lower_connect)((100,140));
606 expand_pbox;
607 enddef;

```



```

608
609 vardef latin.upr =
610   push_pbox_toexpand("latin.upr");
611   latin.upp_base(360);
612   replace_strokep(0)(oldp-(xpart point infinity of oldp,latin_wide_low_v));
613   replace_strokeq(0)(oldq-(1.6,1.6));
614   set_botip(0,4,1);
615   set_boserif(0,4,1);
616   set_boserif(0,5,3);
617
618   push_stroke((point 0.2 of get_stroke(0)){right}..(630,310)..
619     tension 3..(820,latin_wide_low_v),
620     (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
621   replace_strokep(0)(insert_nodes(oldp)(1.95));
622   set_boserif(0,3,3);
623
624   tsu_accent.shift_anchors(ypart olda>vmetric(0.52))

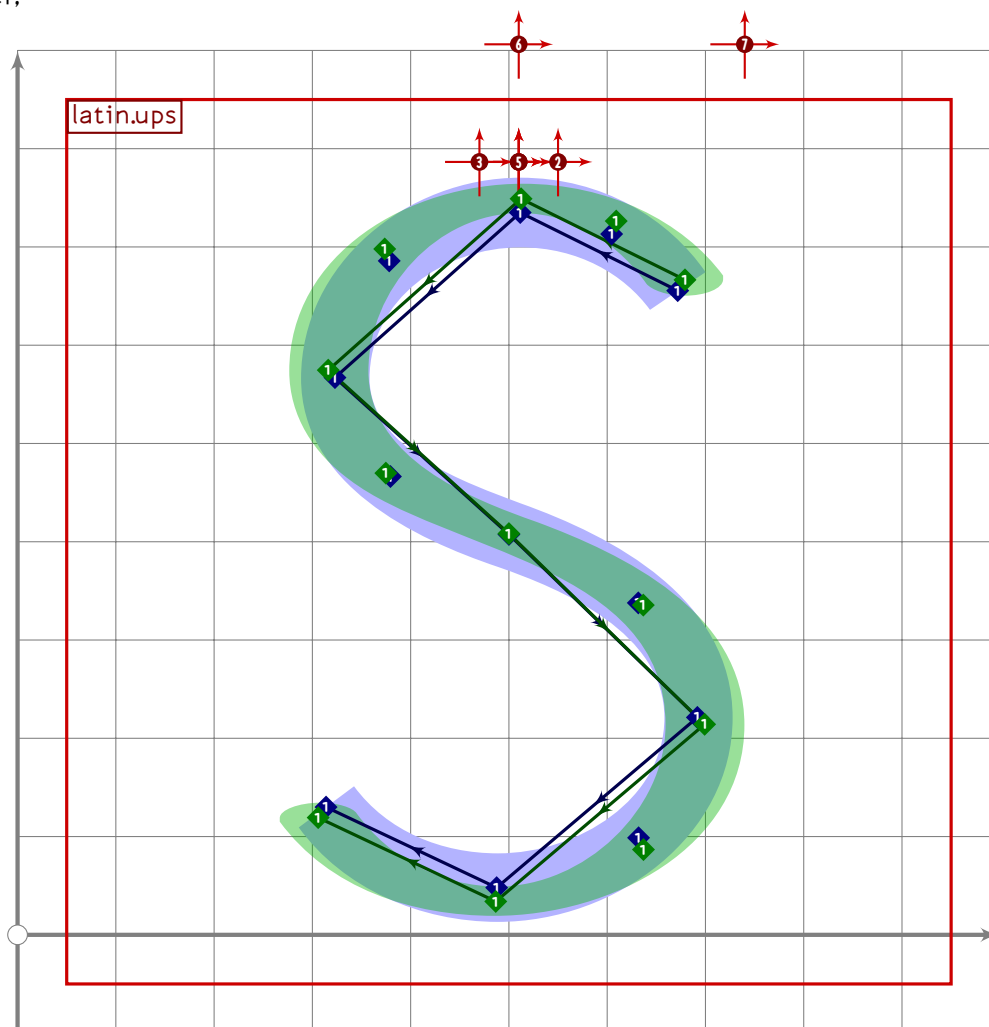
```

U+FF33
tsuku.uniFF33

```

625    (((0,0) transformed tsu_xf.cap_upper_accent)-
626    ((0,0) transformed accent_default[anc_upper]));
627    tsu_accent.shift_anchors(ai=anc_lower_connect)((-200,0));
628    expand_pbox;
629 enddef;

```



```

630
631 vardef latin.ups =
632   push_pbox_toexpand("latin.ups");
633   transform ta,tb;
634   path mycurve;
635
636   mycurve:=(1,0)..(0,1)..(-1,0);
637
638   y2=latin_wide_high_r;
639   y0=y3=vmetric(0.77);
640   y4=vmetric(0.53);
641   y5=y8=vmetric(0.25);
642   y6=latin_wide_low_r;
643
644   0.48[x1,x7]=0.48[x2,x6]=0.48[x3,x5]=x4=500;

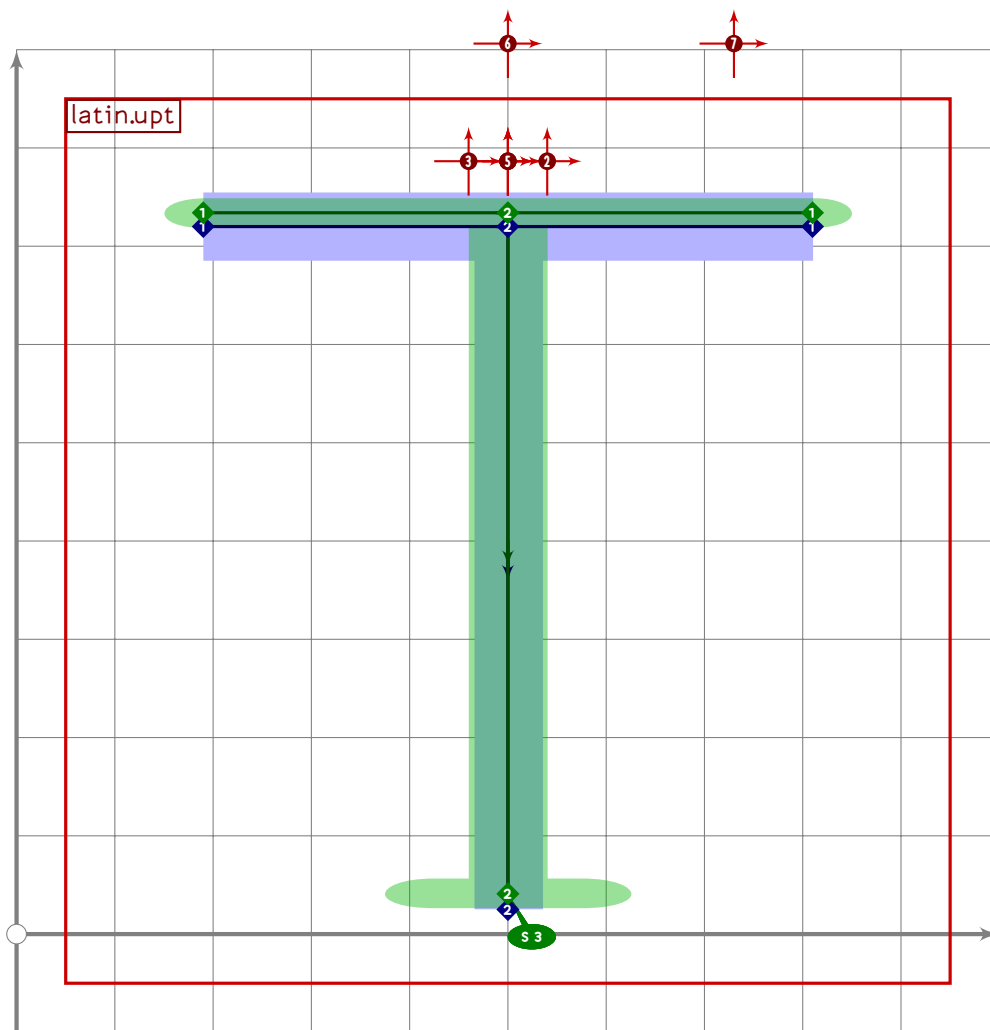
```

LATI


```

645 x5-x1=20;
646 x5-x7=(y2-y6)*0.55;
647
648 (point 0 of mycurve) transformed ta=z0;
649 (point 0.35 of mycurve) transformed ta=z1;
650 (point 1 of mycurve) transformed ta=z2;
651 (point 2 of mycurve) transformed ta=z3;
652 xypart ta=0;
653
654 (point 0 of mycurve) transformed tb=z8;
655 (point 0.35 of mycurve) transformed tb=z7;
656 (point 1 of mycurve) transformed tb=z6;
657 (point 2 of mycurve) transformed tb=z5;
658
659 mycurve:=subpath (0.35,2) of mycurve;
660
661 push_stroke((mycurve transformed ta)..z4..(reverse mycurve transformed tb),
662   (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)
663     -(1.6,1.6));
664
665 tsu_accent.shift_anchors(ypart olda>vmetric(0.52))
666   (((0,0) transformed tsu_xf.cap_upper_accent)-
667     ((0,0) transformed accent_default[anc_upper])+(10,0));
668 expand_pbox;
669 enddef;

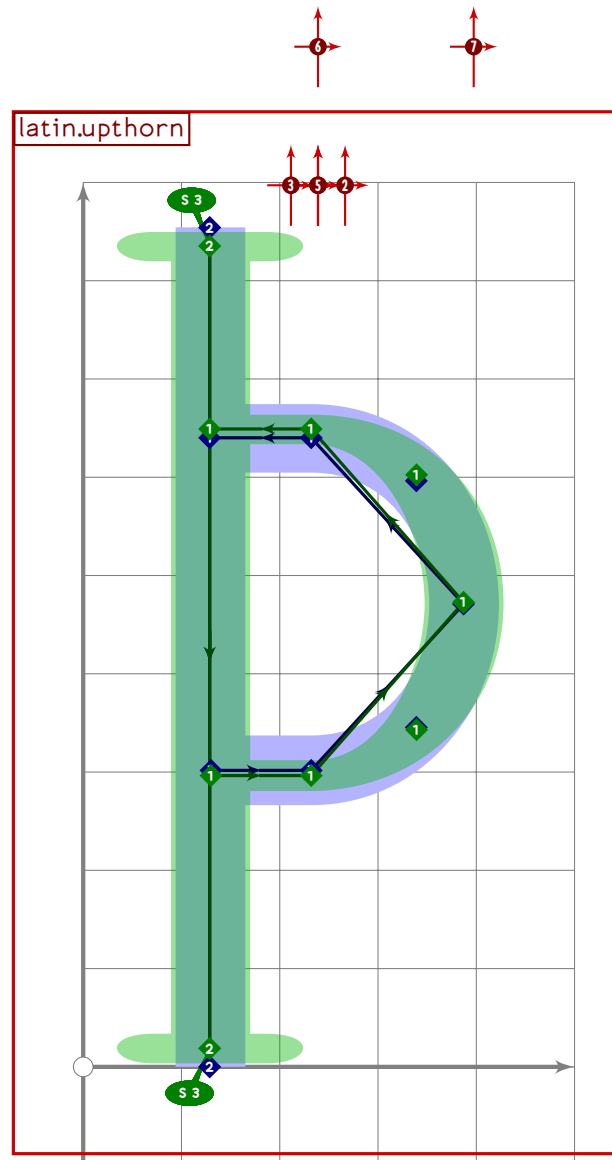
```



```

670
671 vardef latin.upt =
672   push__pbox_toexpand("latin.upt");
673   z1=(190,latin_wide_high_h);
674   z2=(500,latin_wide_high_h);
675   z3=(810,latin_wide_high_h);
676   z4=(500,latin_wide_low_v);
677
678   push__stroke(z1-z3,(1.6,1.6)-(1.6,1.6));
679
680   push__stroke(z2-z4,(1.6,1.6)-(1.6,1.6));
681   set__boserif(0,1,3);
682
683   tsu__accent.shift__anchors(ypart olda>vmetric(0.52))
684     (((0,0) transformed tsu__xf.cap_upper_accent)-
685       ((0,0) transformed accent_default[anc_upper]));
686   expand__pbox;
687 enddef;

```



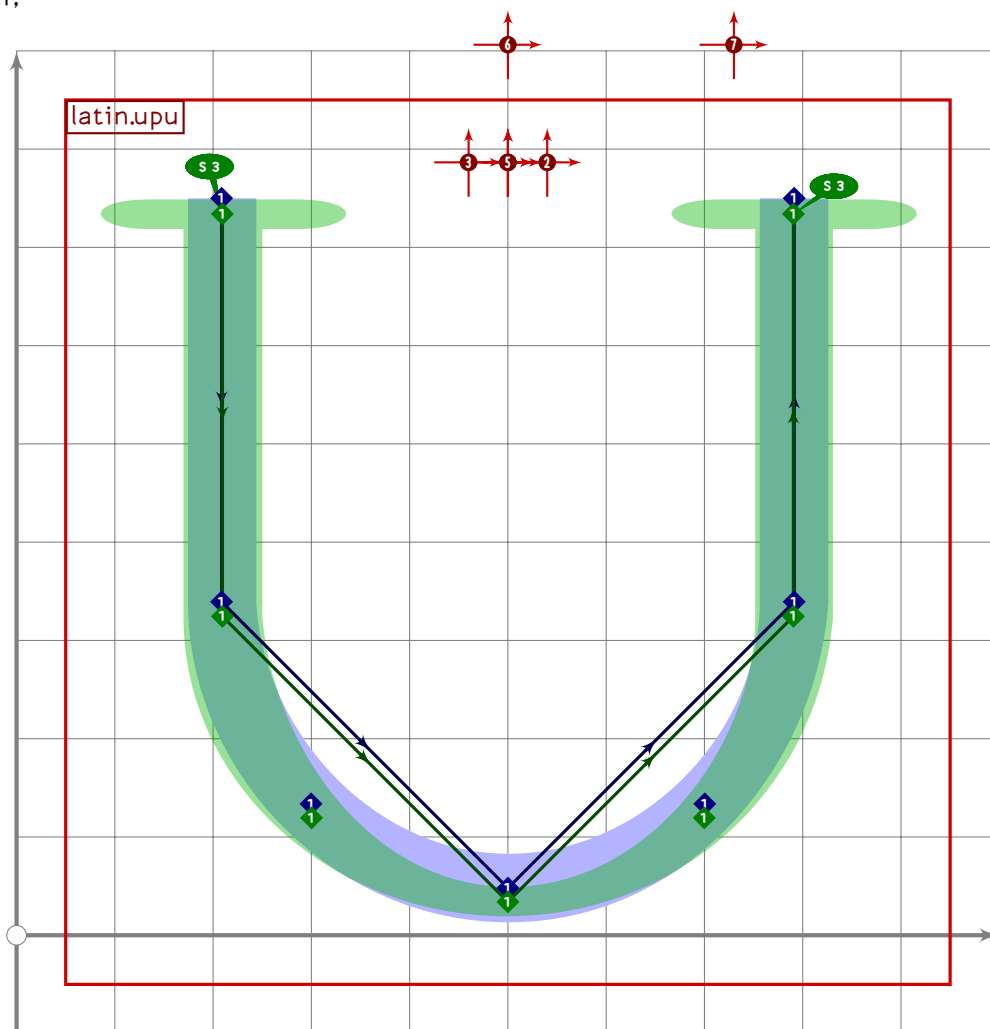
```

688
689 vardef latin.upthorn =
690   push_pbox_toexpand("latin.upthorn");
691   latin.upp_base(375);
692   replace_stroke(0)(oldp
693     shifted (-llcorner oldp) yscaled 0.9
694     shifted ((llcorner oldp)+(0,vmetric(0)-vmetric(0.18))));
695   push_stroke((xpart point infinity of get_stroke(0),latin_wide_high_v)
696     -(xpart point infinity of get_stroke(0),latin_wide_low_v),
697     (1.6,1.6)-(1.6,1.6));
698   set_boserif(0,0,3);
699   set_boserif(0,1,3);
700
701   tsu_accent.shift_anchors(ypart olda>vmetric(0.52))
702     (((0,0) transformed tsu_xf.cap_upper_accent)-
703     ((0,0) transformed accent_default[anc_upper]));

```

U+FF35
tsuku.uniFF35

```
704 expand_pbox;
705 enddef;
```



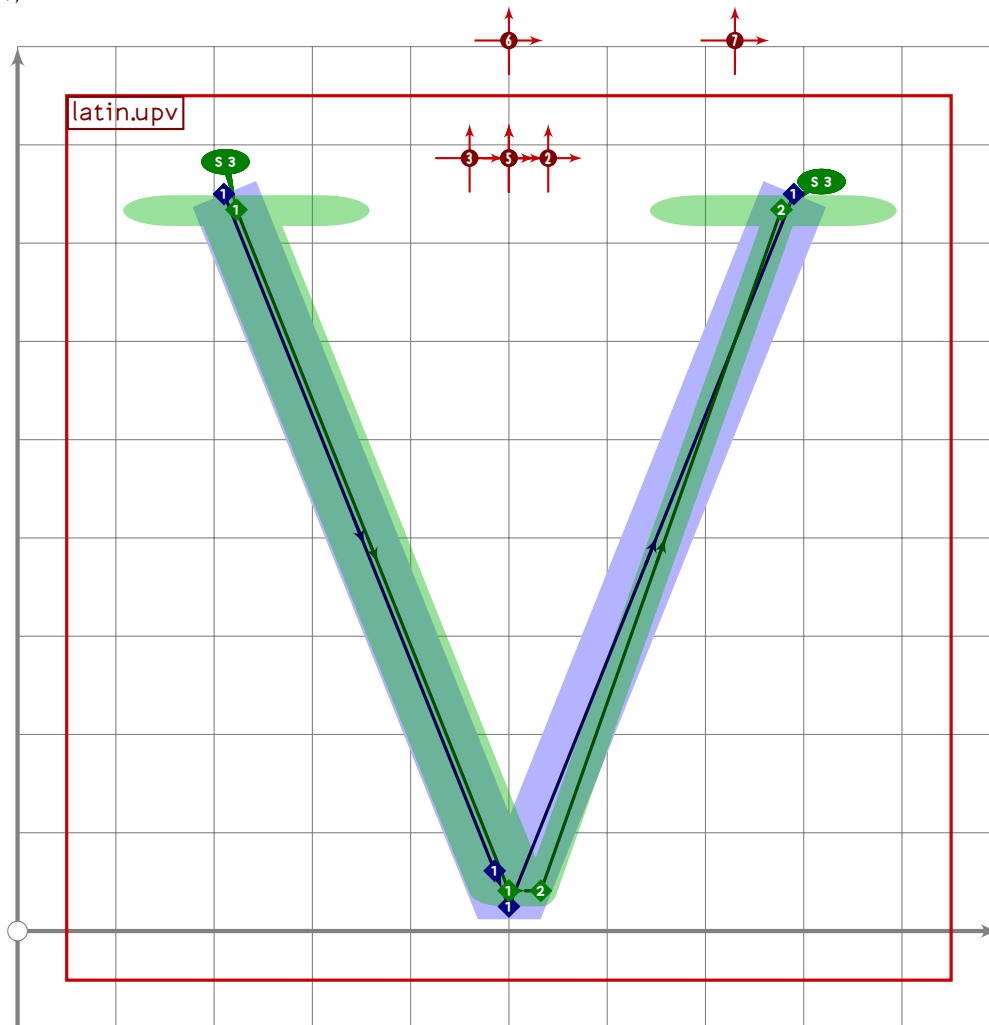
```
706
707 vardef latin.upu =
708   push_pbox_toexpand("latin.upu");
709   x1=x2;
710   x3=500;
711   x4=x5;
712   (x1+x5)/2=x3;
713   (x5-x1)=(y1-y3)*0.83;
714
715   y1=y5=latin_wide_high_v;
716   y2=y4;
717   y2-y3=x3-x2;
718   y3=latin_wide_low_r;
719
720   push_stroke(z1-z2{dir 274}..z3.{dir 86}z4-z5,
721     (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
722   set_boserif(0,0,3);
723   set_boserif(0,4,3);
```

LATI

```

724
725 tsu_accent.shift_anchors(ypart olda>vmetric(0.52))
726 (((0,0) transformed tsu_xf.cap_upper_accent)-
727 ((0,0) transformed accent_default[anc_upper]));
728 expand_pbox;
729 enddef;

```



```

730
731 vardef latin.upv =
732   push_pbox_toexpand("latin.upv");
733   (x1+x3)/2=x2=500;
734
735   y1=y3=latin_wide_high_v;
736   y2=latin_wide_low_v;
737
738   (x3-x1)=(y1-y2)*0.8;
739
740   if do_alteration:
741     push_stroke(z1-z2,(1.6,1.6)-(1.6,1.6));
742     set_boserif(0,0,3);
743

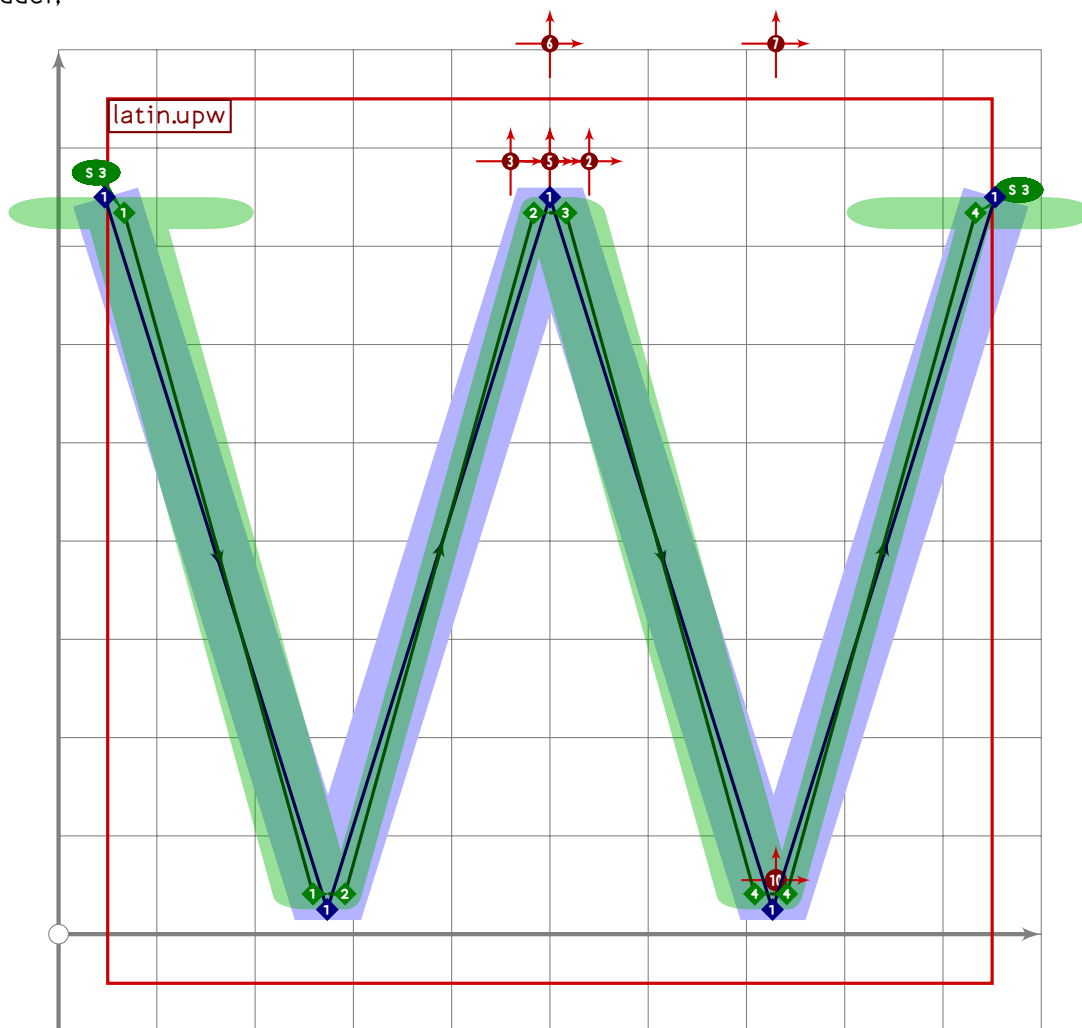
```

U+FF37
tsuku.uniFF37

```

744   push_stroke(z2-(z2+alternate_adjust*right)-z3,
745     (1.6,1.6)-(1.6,1.6)-(1.6,1.6));
746   set_boserif(0,2,3);
747   set_bobrush(0,bralternate);
748   else:
749     push_stroke(z1-(0.95[z1,z2])-z2-z3,
750       (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
751     set_botip(0,2,0);
752     set_boserif(0,0,3);
753     set_boserif(0,3,3);
754   fi;
755
756   tsu_accent.shift_anchors(ypart olda>vmetric(0.52))
757     (((0,0) transformed tsu_xf.cap_upper_accent)-
758     ((0,0) transformed accent_default[anc_upper]));
759   expand_pbox;
760 enddef;

```



```

761
762 vardef latin.upw =
763   push_pbox_toexpand("latin.upw");

```

LATI

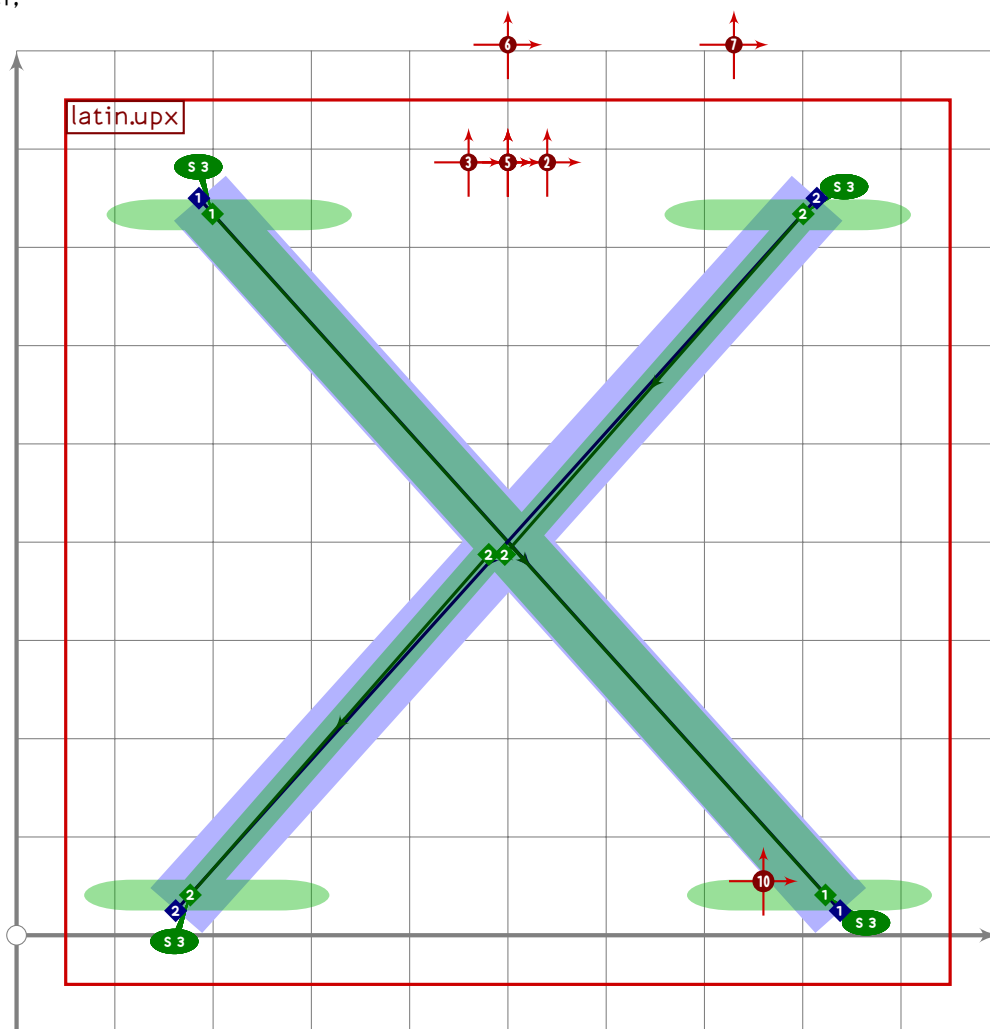
```

764 if do_alteration:
765     (x1+x5)/2=(x2+x4)/2=x3=500-alternate_adjust/2;
766     (x3-x2)=(x2-x1);
767
768     y1=y3=y5=latin_wide_high_v;
769     y2=y4=latin_wide_low_v;
770
771     (x5-x1)=(y1-y2)*1.25-(3*alternate_adjust);
772
773     push_stroke((z1-z2) shifted (alternate_adjust*left),
774         (1.6,1.6)-(1.6,1.6));
775     set_boserif(0,0,3);
776
777     push_stroke((z2+alternate_adjust*left)-z2-
778         z3-(z3+alternate_adjust*right),
779         (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
780     set_bobrush(0,bralternate);
781
782     push_stroke((z3-z4) shifted (alternate_adjust*right),
783         (1.6,1.6)-(1.6,1.6));
784
785     push_stroke((z4+alternate_adjust*right)-
786         ((z4-z5) shifted (alternate_adjust*right*2)),
787         (1.6,1.6)-(1.6,1.6)-(1.6,1.6));
788     set_boserif(0,2,3);
789     set_bobrush(0,bralternate);
790 else:
791     (x1+x5)/2=(x2+x4)/2=x3=500;
792     (x3-x2)=(x2-x1);
793
794     y1=y3=y5=latin_wide_high_v;
795     y2=y4=latin_wide_low_v;
796
797     (x5-x1)=(y1-y2)*1.25;
798
799     push_stroke(z1-z2-z3-z4-z5,
800         (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
801     set_botip(0,1,0);
802     set_botip(0,2,0);
803     set_botip(0,3,0);
804     set_boserif(0,0,3);
805     set_boserif(0,4,3);
806 fi;
807
808 tsu_accent.shift_anchors(ypart olda>vmetric(0.52))
809     (((0,0) transformed tsu_xf.cap_upper_accent)-
810     ((0,0) transformed accent_default[anc_upper]));
811 tsu_accent.shift_anchors(ai=anc_lower_connect)((230,0));

```

U+FF38
tsuku.uniFF38

```
812 expand_pbox;
813 enddef;
```



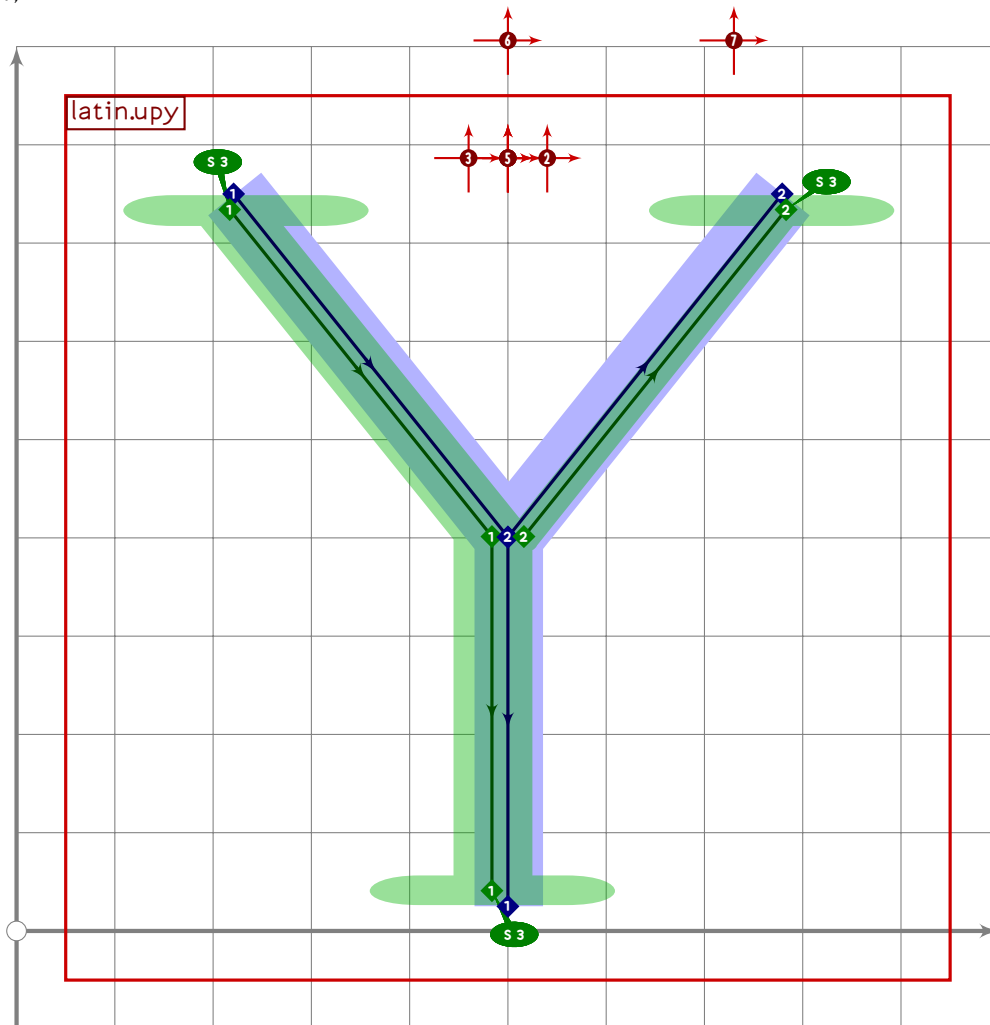
```
814
815 vardef latin.upx =
816   push_pbox_toexpand("latin.upx");
817   (x1+x3)/2=500;
818   (x2+x4)/2=500;
819   (x2+x3-x1-x4)=((y1-y2)*0.9)*2;
820   (x3-x1)=(x2-x4)*0.93;
821
822   y1=y3=latin_wide_high_v;
823   y2=y4=latin_wide_low_v;
824
825   push_stroke(z1-z2,(1.6,1.6)-(1.6,1.6));
826   set_boserif(0,0,3);
827   set_boserif(0,1,3);
828
829   if do_alteration:
830     push_stroke(z3-(0.5[z3,z4]+alternate_adjust*right/4)
831               -(0.5[z3,z4]+alternate_adjust*left/4)-z4,
```

LATI


```

832     (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
833     set_boserif(0,0,3);
834     set_boserif(0,3,3);
835   else:
836     push_stroke(z3-z4,(1.6,1.6)-(1.6,1.6));
837     set_boserif(0,0,3);
838     set_boserif(0,1,3);
839   fi;
840   set_bobrush(0,bralternate);
841
842   tsu_accent.shift_anchors(ypart olda>vmetric(0.52))
843     (((0,0) transformed tsu_xf.cap_upper_accent)-
844      ((0,0) transformed accent_default[anc_upper]));
845   tsu_accent.shift_anchors(ai=anc_lower_connect)((260,0));
846   expand_pbox;
847 enddef;

```



```

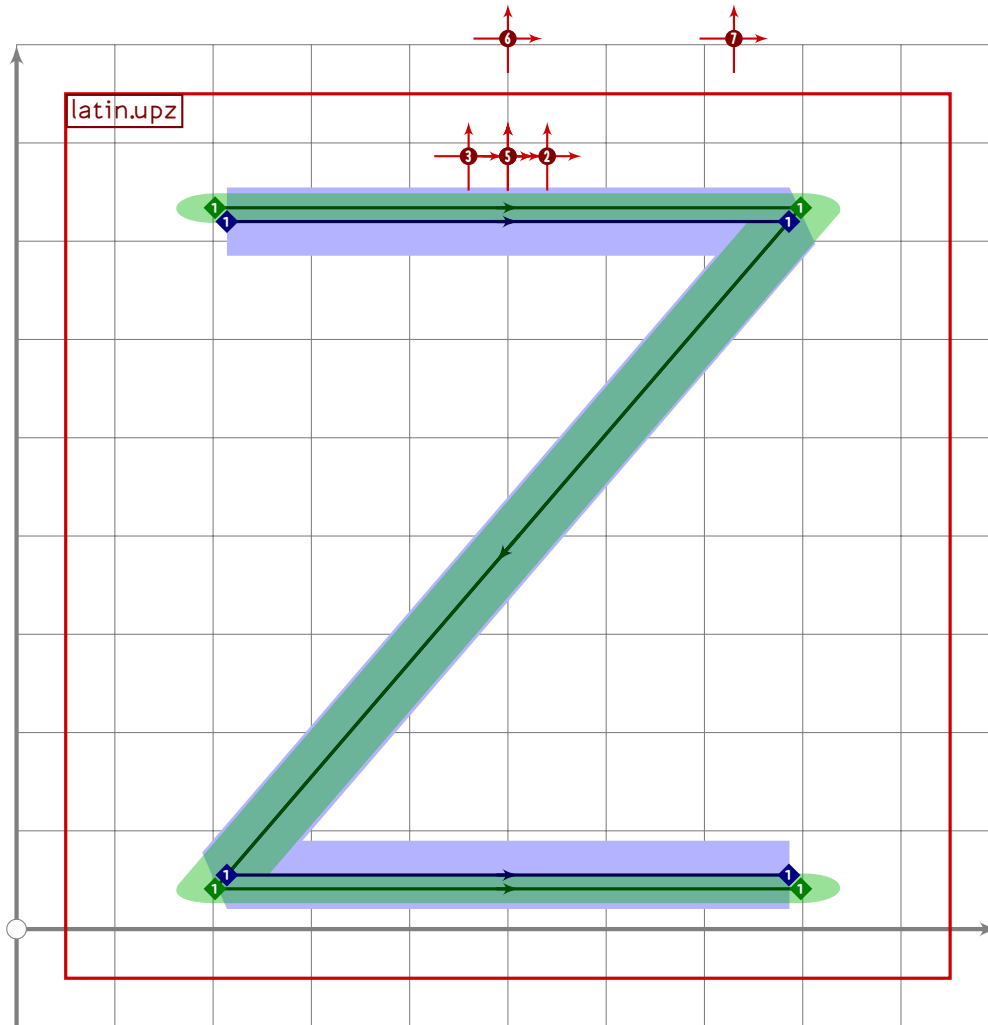
848
849 vardef latin.upy =
850   push_pbox_toexpand("latin.upy");
851   (x3+x1)/2=x2=x4=if do_alteration: 500-alternate_adjust/2 else: 500 fi;

```

```

852 (x3-x1)=0.77*(y1-y4);
853
854 y1=y3=latin_wide_high_v;
855 y2=vmetric(0.52);
856 y4=latin_wide_low_v;
857
858 push_stroke(z1-z2-z4,(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
859 set_botip(0,1,0);
860 set_boserif(0,0,3);
861 set_boserif(0,2,3);
862
863 if do_alteration:
864     push_stroke((z2-z3) shifted (alternate_adjust*right),
865         (1.6,1.6)-(1.6,1.6));
866 else:
867     push_stroke(z2-z3,(1.6,1.6)-(1.6,1.6));
868 fi;
869 set_boserif(0,1,3);
870 set_bobrush(0,bralternate);
871
872 tsu_accent.shift_anchors(y part olda>vmetric(0.52))
873     (((0,0) transformed tsu_xf.cap_upper_accent)-
874     ((0,0) transformed accent_default[anc_upper]));
875 expand_pbox;
876 endif;

```



```

877
878 vardef latin.upz =
879   push_pbox_toexpand("latin.upz");
880   y1=y2=latin_wide_high_h;
881   y3=y4=latin_wide_low_h;
882
883   x1=x3;
884   x2=x4;
885   (x1+x2)/2=500;
886   (x2-x1)=(y1-y3)*0.86;
887
888   push_stroke(z1-z2-z3-z4,(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
889   set_botip(0,1,0);
890   set_botip(0,2,0);
891
892   tsu_accent.shift_anchors(ypart olda>vmetric(0.52))
893     (((0,0) transformed tsu_xf.cap_upper_accent)-
894     ((0,0) transformed accent_default[anc_upper]));
895   expand_pbox;
896 enddef;

```

LATI

```

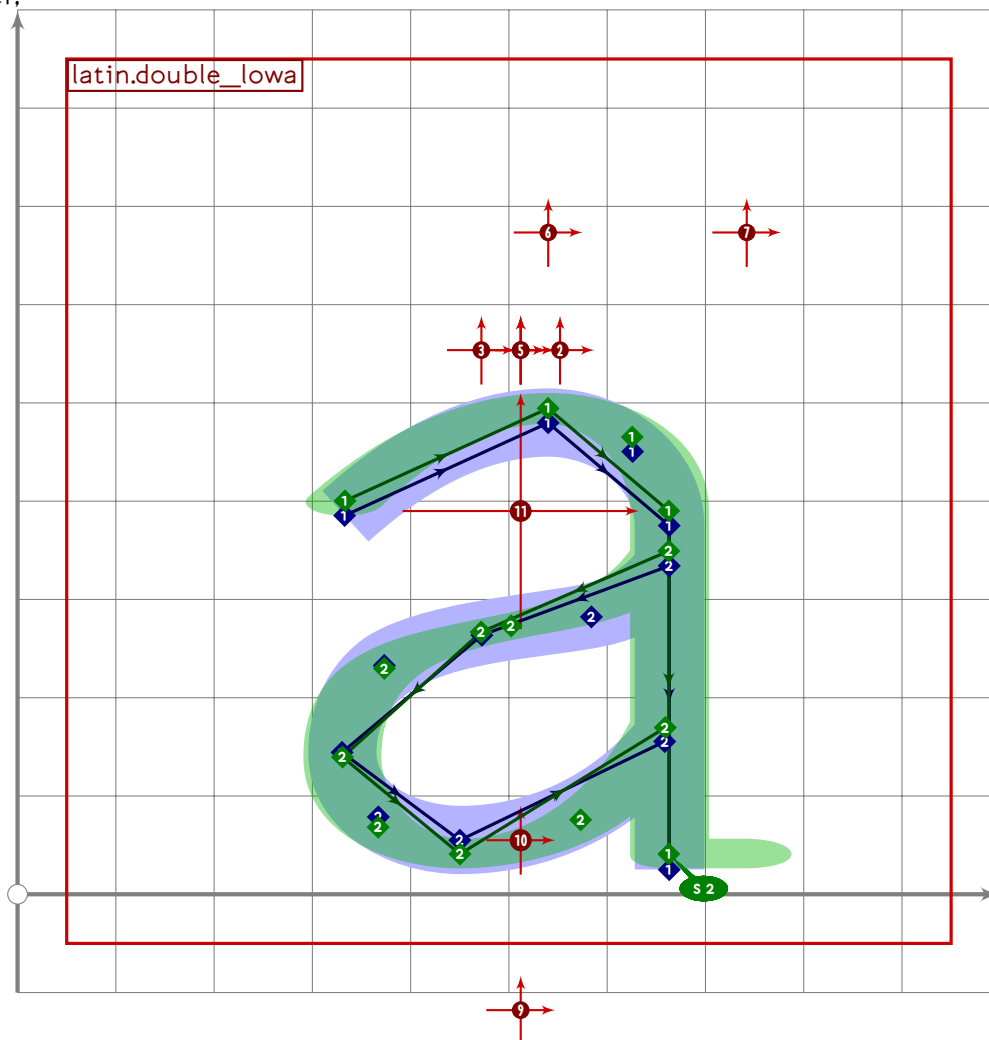
897
898
899
900 vardef latin.double_lowa =
901   push_pbox_toexpand("latin.double_lowa");
902   x1=(-0.12)[x6,x3];
903   x3=x4=x5=x8;
904   0.51[x6,x3]=500;
905   x3-x1=0.82*(y2-y4);
906   x2=0.63[x6,x3];
907   x7=0.36[x6,x3];
908
909   y1=0.7[y4,y2];
910   y3=0.77[y4,y2];
911   y2=latin_wide_xheight_r;
912   y4=latin_wide_low_v;
913   y5=0.68[y4,y2];
914   y6=0.32[y7,y5];
915   y7=latin_wide_low_h;
916   y8=0.3[y4,y2];
917
918   push_stroke(z1{curl 0.2}..z2{right}..z3{down}..z4,
919     (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
920   replace_strokep(0)(subpath (0.2,3) of oldp);
921   set_boserif(0,3,if do_italic_hook: 11 else: 2 fi);
922
923   push_stroke(z5{dir 240}..z6{down}..z7{right}..z8,
924     (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
925   replace_strokep(0)(subpath (0,2.97) of oldp);
926   replace_strokep(0)(insert_nodes(oldp)(0.5));
927
928   tsu_accent.shift_anchors(true)((12,0));
929   tsu_accent.shift_anchors(ai=anc_ring)((28,0));
930   expand_pbox;
931 enddef;
932
933 vardef latin.single_lowa =
934   push_pbox_toexpand("latin.single_lowa");
935   (x1+x4)/2=520;
936   (x1-x4)=(y3-y1)*0.81;
937   x1=x6-8;
938   x3=0.4[x1,x4];
939   x5=0.36[x4,x1];
940
941   y1=latin_wide_low_v;
942   y3=latin_wide_xheight_h;
943   y4=0.47[y5,y3];
944   y5=latin_wide_low_h;

```

```

945 y6=0.37[y1,y3];
946
947 z7=(x1,y3);
948 z2=0.3[z7,0.5[z3,z1]];
949
950 push_stroke(interpath(0.5)
951   (z1{up}{.}{up}{z7{left}{.}{left}{z3.{down}{z4.{right}{z5..z6,
952     z1{up}{.}{z2.{left}{z3.{down}{z4.{right}{z5..z6),
953     (1.3,1.3)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-
954     (1.6,1.6)-(1.6,1.6)-(1.6,1.6));
955 replace_strokep(0)(reverse(insert_nodes(oldp)(0.5)));
956 set_boserif(0,6,if do_italic_hook: 11 else: 2 fi);
957 set_botip(0,4,1);
958 expand_pbox;
959 enddef;

```

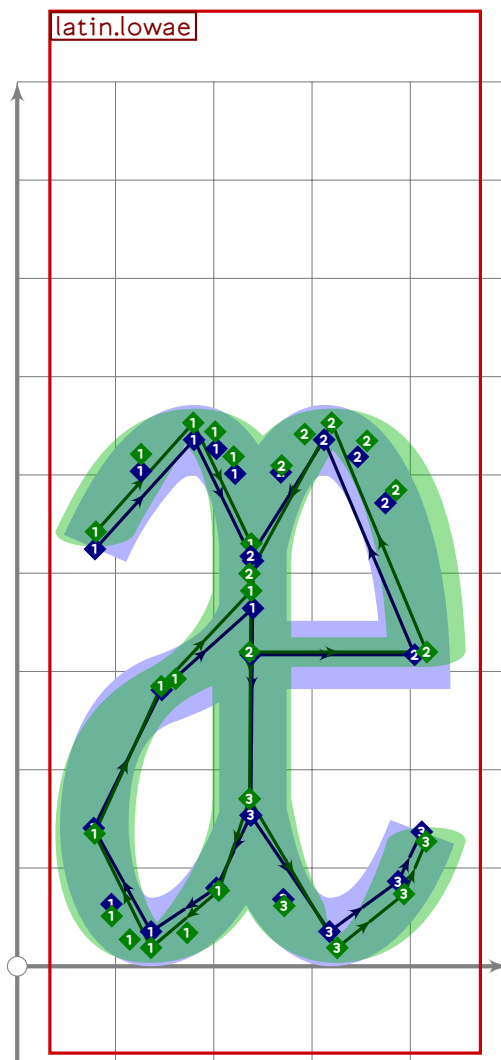


LATI

```

960
961 vardef latin.low_a =
962   latin.double_low_a;
963 enddef;

```



```

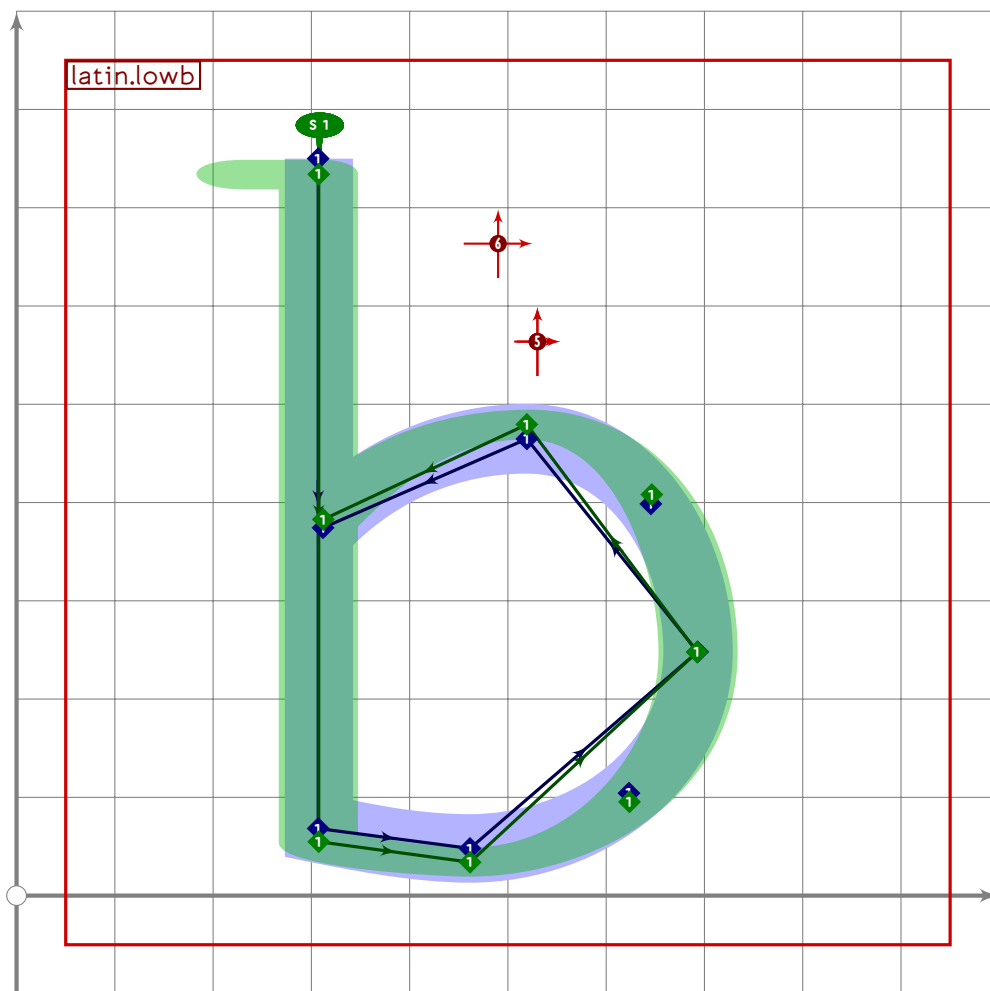
964
965 vardef latin.lowae =
966   push_pbox_toexpand("latin.lowae");
967   begingroup
968     save saved_sp;
969     save astem,abowl,astroke,estroke,etop,ebot;
970     path astem,abowl,astroke,estroke,etop,ebot;
971     saved_sp:=sp;
972
973     latin.double_lowae;
974     set_boserif(-1,3,whatever);
975     astem:=get_strokep(-1);
976     abowl:=get_strokep(0);
977
978     numeric x[],y[];
979     latin.lowe;
980     estroke:=get_strokep(0);
981
982     for i=saved_sp upto sp-1:

```

```

983     obstacktype[i]:=otnull;
984 endfor;
985
986 astroke:=(subpath (0,2) of astem)-reverse abowl;
987
988 numeric x[],y[];
989 z1=point 3 of astroke;
990 path xbp;
991 xbp=estroke shifted (0,ypart ((llcorner astroke)-(llcorner estroke)));
992 x2=xpart (xbp intersectionpoint ((470,y1)-(0,y1)));
993 astroke:=astroke shifted (470-x1,0);
994
995 estroke:=estroke shifted (470-x2,0);
996 xbp:=xbp shifted (470-x2,0);
997
998 ebot:=subpath (xpart (xbp intersectiontimes ((500,y1)-(0,y1))),
999             infinity) of xbp;
1000 ebot:=insert_nodes(ebot)((length ebot)-0.3);
1001
1002 etop:=
1003     subpath (ypart (((470,1000)-(470,0))
1004             intersectiontimes (subpath (0,1) of estroke)),
1005             1+ypart (((470,1000)-(470,0))
1006             intersectiontimes (subpath (1,infinity) of estroke)))
1007     of estroke;
1008
1009 astroke:=insert_nodes(astroke)(3,4);
1010
1011 push_stroke(astroke,
1012     (1.6,1.6) for i=1 upto length astroke: -(1.6,1.6) endfor);
1013 push_stroke(etop,
1014     (1.6,1.6) for i=1 upto length etop: -(1.6,1.6) endfor);
1015 set_botip(0,1,1);
1016 push_stroke(ebot,
1017     (1.6,1.6) for i=1 upto length ebot: -(1.6,1.6) endfor);
1018 endgroup;
1019 expand_pbox;
1020 enddef;

```



```

1021
1022 vardef latin.lowb =
1023   push_pbox_toexpand("latin.lowb");
1024   (x1+x4)/2=500;
1025   (x4-x1)=(y1-y3)*0.55;
1026   x2=x1=x6;
1027   x3=0.4[x2,x4];
1028   x5=0.55[x2,x4];
1029
1030   y1=latin_wide_high_v;
1031   y2=latin_wide_lc_baselif;
1032   y3=latin_wide_low_r;
1033   y4=0.48[y3,y5];
1034   y5=latin_wide_xheight_h;
1035   y6=0.77[y3,y5];
1036
1037   push_stroke(z1-z2{curl 0.05}{right}z3.{up}z4.{left}z5..z6,
1038     (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
1039   replace_strokep(0)(subpath (0,4,97) of oldp);
1040   set_botip(0,1);
1041   if not do_italic_hook: set_boserif(0,0,1); fi;

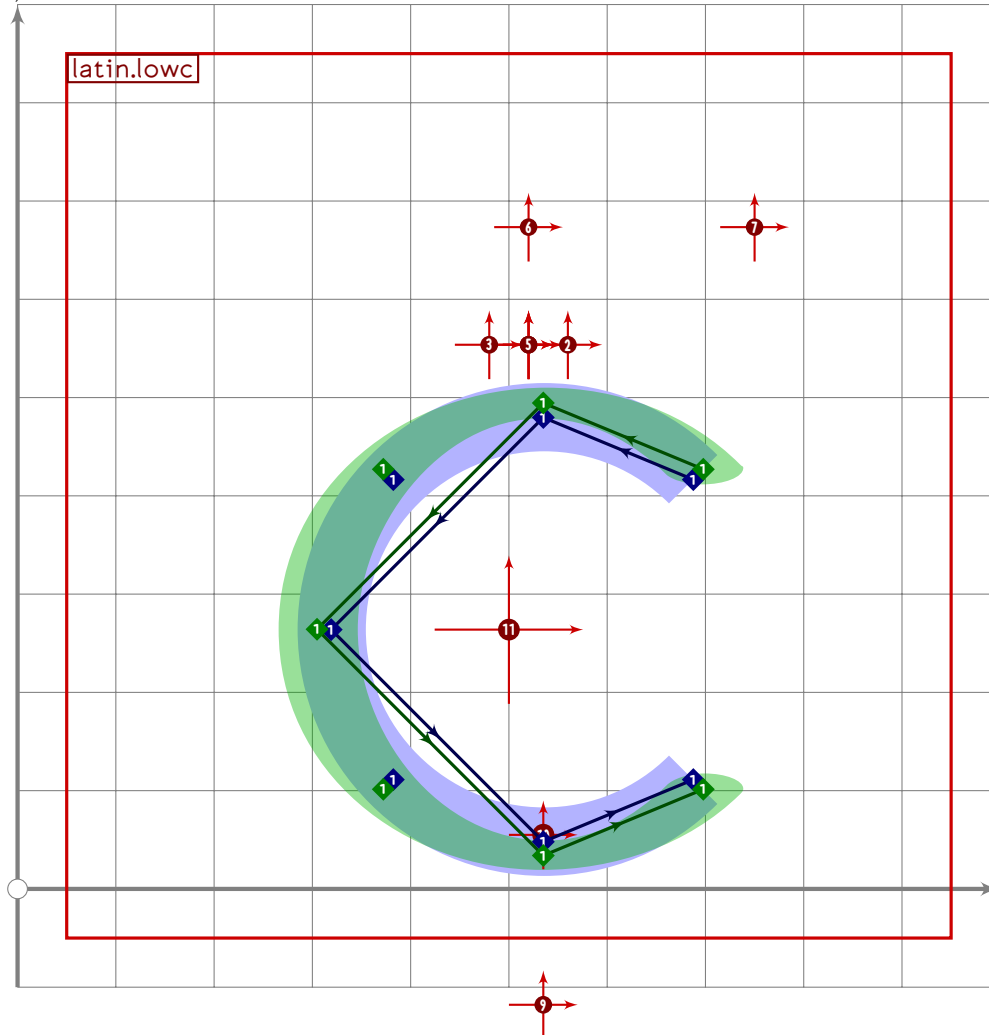
```



```

1042
1043   push_anchor(anc_wide,identity xscaled 0.9
1044     transformed accent_default[anc_wide] shifted (30,10));
1045   push_anchor(anc_tilde,identity xscaled 0.8
1046     transformed accent_default[anc_tilde] shifted (30,10));
1047   push_anchor(anc_ring,accent_default[anc_ring] shifted (-10,-10));
1048   expand_pbox;
1049 enddef;

```



```

1050
1051 vardef latin.lowc =
1052   push_pbox_toexpand("latin.lowc");
1053   push_stroke((subpath (0.5,3.5) of ((1,0)..(0,1)..(-1,0)..(0,-1)..cycle))
1054     scaled ((latin_wide_xheight_r-latin_wide_low_r)/2)
1055     shifted ((xpart centre_pt,(latin_wide_xheight_r+latin_wide_low_r)/2)
1056       +(35,0)),
1057     (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
1058   tsu_accent.shift_anchors(ypart olda>vmetric(0.52))((20,0));
1059   tsu_accent.shift_anchors(ypart olda<=vmetric(0.52))((35,0));
1060   push_anchor(anc_centre,identity
1061     scaled ((latin_wide_xheight_r-latin_wide_low_r)/200)

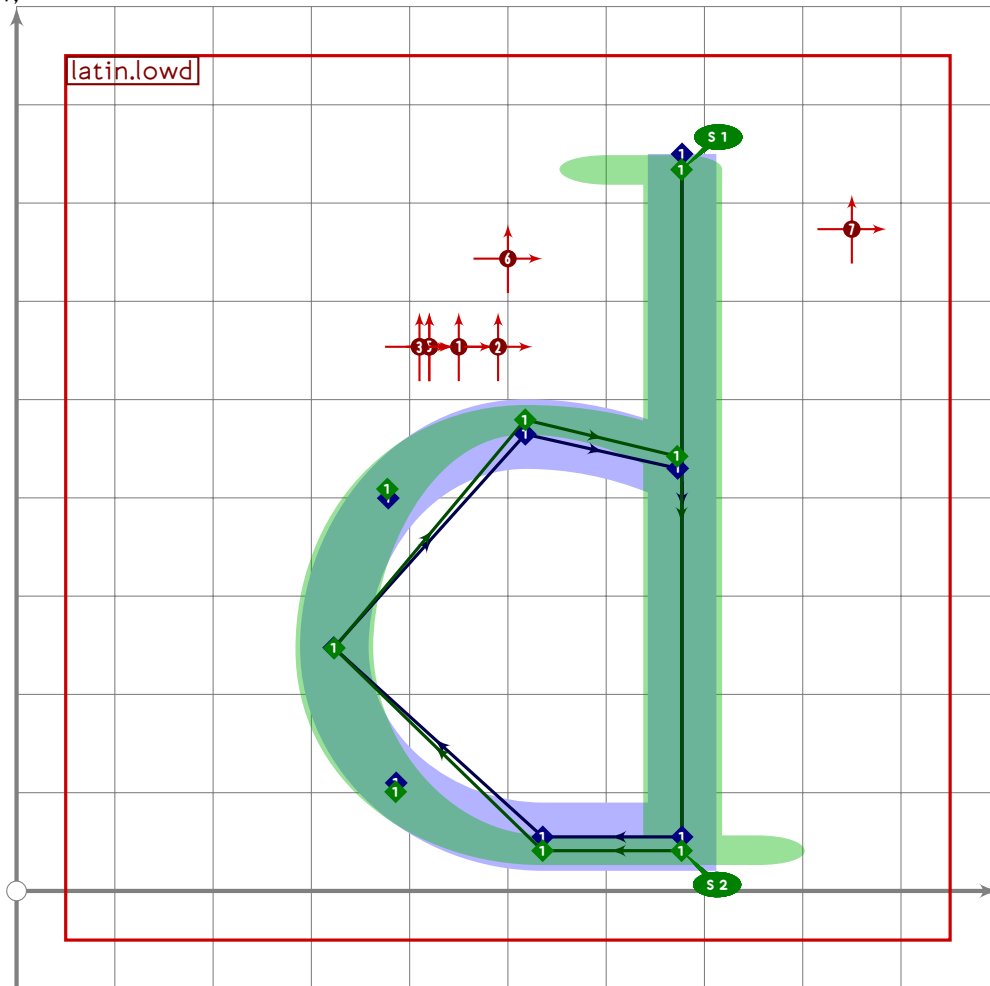
```

U+FF44
tsuku.uniFF44

```

1062   shifted (xpart centre_pt,(latin_wide_xheight_r+latin_wide_low_r)/2));
1063   expand_pbox;
1064 enddef;

```



```

1065
1066 vardef latin.lowd =
1067   push_pbox_toexpand("latin.lowd");
1068   (x1+x4)/2=500;
1069   (x1-x4)=(y1-y3)*0.51;
1070   x2=x1=x6;
1071   x3=0.4[x2,x4];
1072   x5=0.45[x2,x4];
1073
1074   y1=latin_wide_high_v;
1075   y2=y3=latin_wide_low_h;
1076   y4=0.47[y3,y5];
1077   y5=latin_wide_xheight_h;
1078   y6=0.91[y3,y5];
1079
1080   push_stroke(z1-z2{left}..{left}z3.{up}z4.{right}z5..z6,
1081     (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-
1082     (1.6,1.6)-(1.6,1.6));

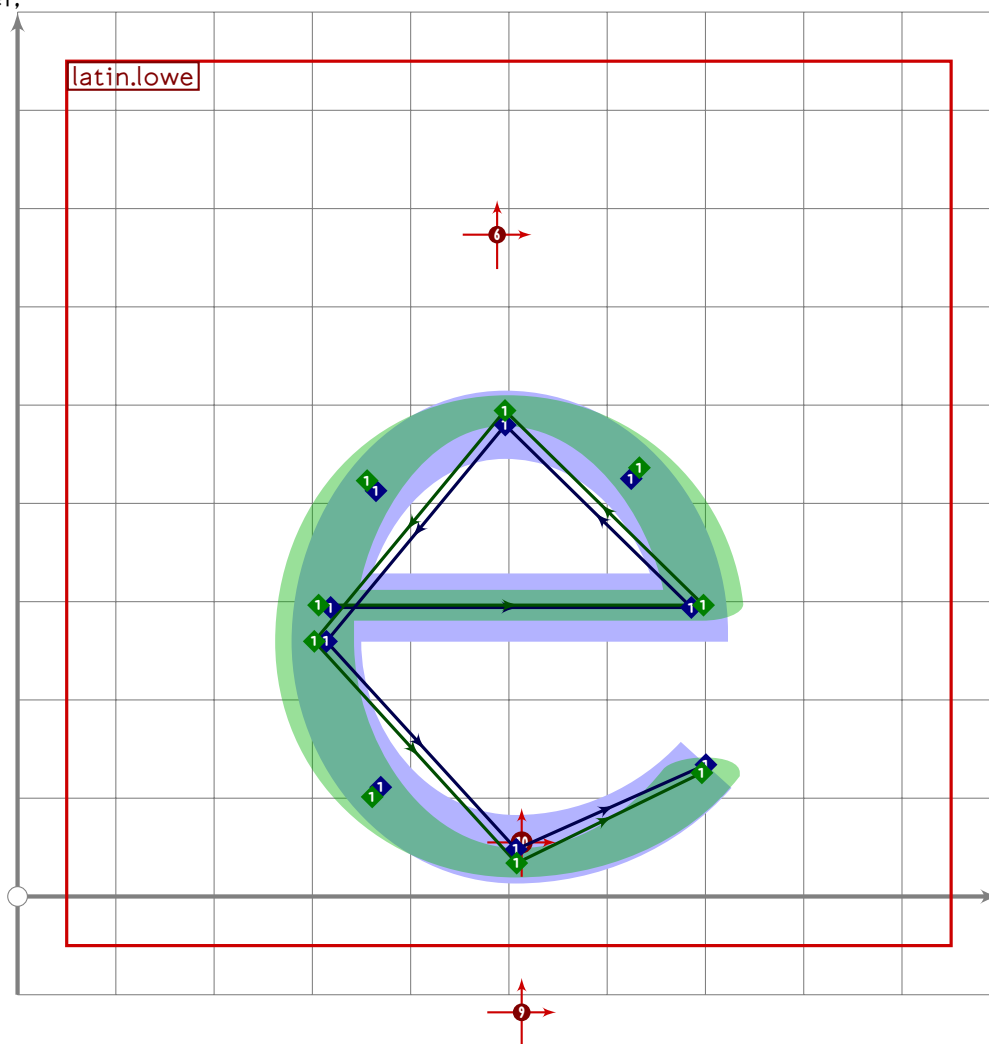
```

LATI

```

1083 replace_strokep(0)(subpath (0,4,97) of oldp);
1084 set_botip(0,1,1);
1085 if not do_italic_hook: set_boserif(0,0,1); fi;
1086 set_boserif(0,1,2);
1087
1088 push_anchor(anc_wide,identity xscaled 0.9
1089   transformed accent_default[anc_wide] shifted (-30,0));
1090 push_anchor(anc_tilde,identity xscaled 0.8
1091   transformed accent_default[anc_tilde] shifted (-30,0));
1092
1093 tsu_accent.shift_anchors(ypart olda>vmetric(0.52))((-50,0));
1094 push_anchor(anc_ring,accent_default[anc_ring] shifted (0,-30));
1095 push_anchor(anc_caron_comma,
1096   accent_default[anc_caron_comma] shifted (120,0));
1097 expand_pbox;
1098 enddef;

```



```

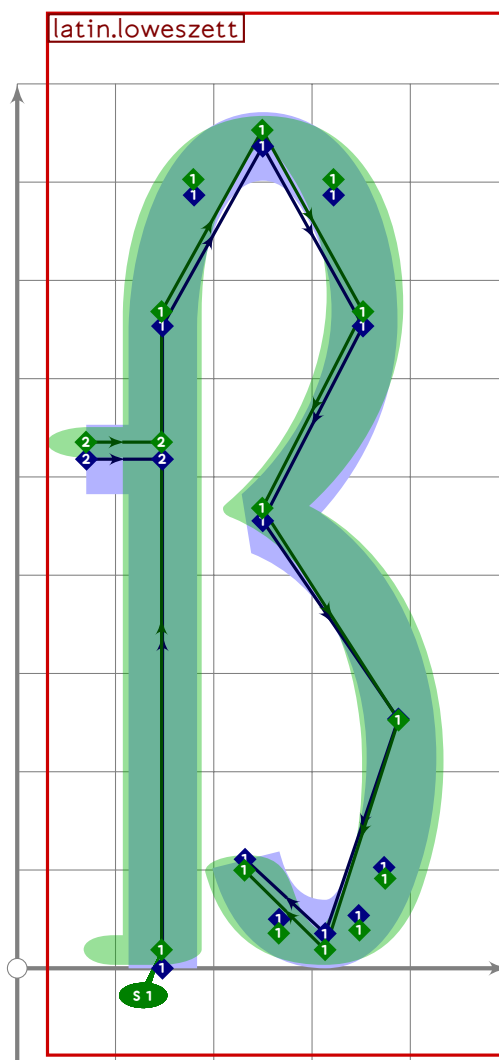
1099
1100 vardef latin.lowe =
1101   push_pbox_toexpand("latin.lowe");
1102   y2=0.57[y5,y3];

```

```

1103 y3=latin_wide_xheight_r;
1104 y4=0.49[y5,y3];
1105 y5=latin_wide_low_r;
1106 y6=0.35[y5,y2];
1107
1108 (x2+x4)/2=500;
1109 (x2-x4)=0.86*(y3-y5);
1110 x3=0.49[x4,x2];
1111 x5=0.52[x4,x2];
1112 x6=1.04[x4,x2]-(if sharp_corners: 0 else: (mbrush_width/3) fi);
1113
1114 push_stroke(z2{curl 0.7}..z3{left}..z4{down}..z5{right}..z6,
1115   (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
1116 z1=get_stroke(0) intersectionpoint ((z2+(-1000,0))-z2+(-10,0));
1117 replace_stroke(0)((z1+2*right)-oldp);
1118 set_botip(0,1);
1119
1120 tsu_accent.shift_anchors(ypart olda<vmetric(0.05))((-13,0));
1121 tsu_accent.shift_anchors(ai=anc_ring)((-12,0));
1122 expand_pbox;
1123 enddef;

```



```

1124
1125 vardef latin.loweszett =
1126   push_pbox_toexpand("latin.loweszett");
1127   (x1+x6)/2=510;
1128   (x6-x1)=3*(y3-y2);
1129   x2=x1;
1130   x3=x5=(x1+x4)/2;
1131   x4=0.85[x1,x6];
1132   x7=0.69[x1,x6];
1133   x8=0.35[x1,x6];
1134
1135   y1=latin_wide_low_v;
1136   y2=y4=0.52[y5,y3];
1137   y3=latin_wide_high_r;
1138   y5=0.87[y7,latin_wide_xheight_h];
1139   y6=0.52[y7,y5];
1140   y7=latin_wide_low_h;
1141   y8=0.18[y7,y5];
1142

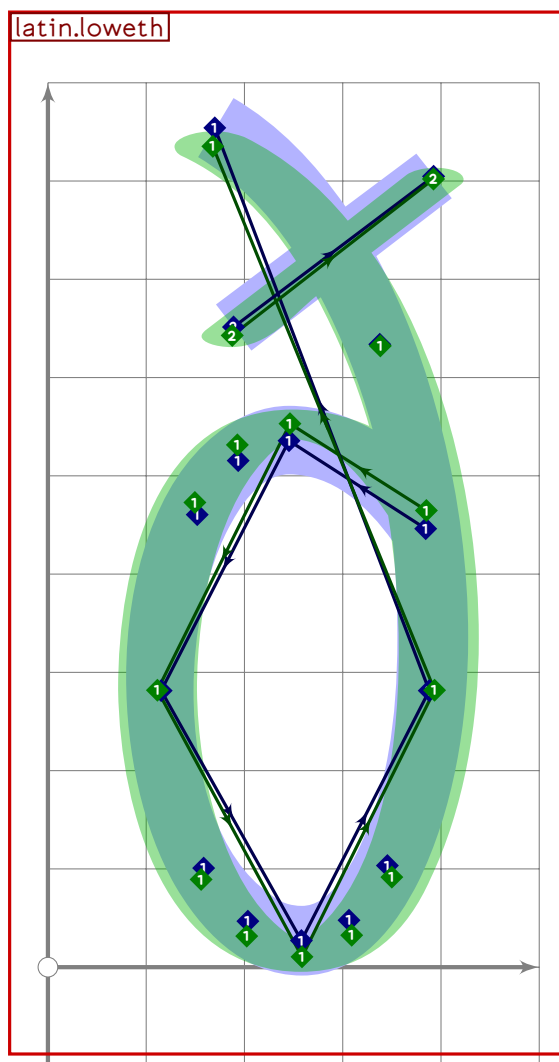
```

U+00F0
tsuku.eth

```

1143 push_stroke(z1-z2{dir 88}..z3{right}..z4..{dir 200}z5{dir 350}..
1144     z6..z7{left}..z8{dir 120},
1145     (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-
1146     (1.6,1.6)-(1.6,1.6)-(1.6,1.6));
1147 set_botip(0,4,0);
1148 set_boserif(0,0,1);
1149
1150 push_stroke((x1-150,latin_wide_xheight_h)-(x1,latin_wide_xheight_h),
1151     (1.6,1.6)-(1.6,1.6));
1152 expand_pbox;
1153 enddef;

```



LATI

```

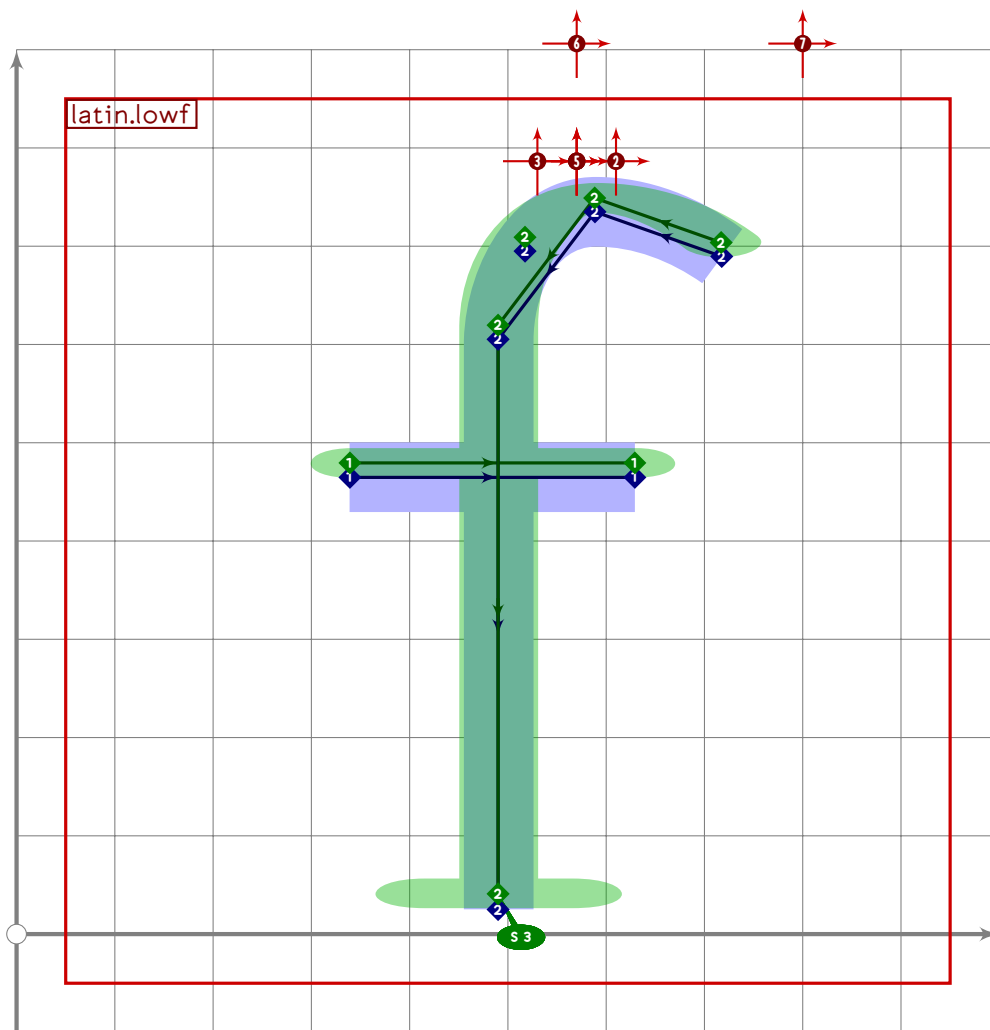
1154
1155 vardef latin.loweth =
1156   push_pbox_toexpand("latin.loweth");
1157   (x3+x5)/2=(x2+x4)/2=510;
1158   x4-x2=20;
1159   (x5-x3)=(y2-y4);
1160   x1=1.3[x3,x5];
1161   x6=0.2[x3,x5];

```

```

1162
1163 y1=0.25[y4,y2];
1164 y3=y5=0.5[y4,y2];
1165 y2=latin_wide_xheight_r;
1166 y4=latin_wide_low_r;
1167 y6=latin_wide_high_v;
1168
1169 push_stroke(z1..z2{left}..z3..z4{right}..z5..{curl 0.6}z6,
1170 (1.6,1.6)–(1.6,1.6)–(1.6,1.6)–(1.6,1.6)–(1.6,1.6)–(1.6,1.6));
1171 replace_strokep(0)(subpath (xpart ((subpath (0,1) of oldp)
1172 intersectiontimes
1173 (subpath (2,infinity) of oldp)),infinity) of oldp);
1174
1175 z7=point 4.67 of get_strokep(0);
1176 z8=z7+whatever*dir 202;
1177 x8=0.27[x3,x5];
1178 z9=2[z8,z7];
1179
1180 push_stroke(z8–z9,(1.5,1.5)–(1.5,1.5));
1181 set_bosize(0)(85);
1182 expand_pbox;
1183 endif;

```



```

1184
1185 vardef latin.lowf =
1186   push_pbox_toexpand("latin.lowf");
1187   (x2-x1)=290;
1188   x5=x6=490=0.52[x1,x2];
1189   x3-x5=2*(y4-y5);
1190   x4=0.38[x5,x3];
1191
1192   y1=y2=latin_wide_xheight_h;
1193   y5=0.52[y2,y4];
1194   y3=0.73[y2,y4];
1195   y4=latin_wide_high_r;
1196   y6=latin_wide_low_v;
1197
1198   push_stroke(z1-z2,(1.6,1.6)-(1.6,1.6));
1199
1200   push_stroke(z3{curl 0.6}..z4{left}..{dir 268}z5{down}-z6,
1201     (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
1202   replace_strokep(0)(subpath (0.23,3) of oldp);
1203   set_boserif(0,3,3);

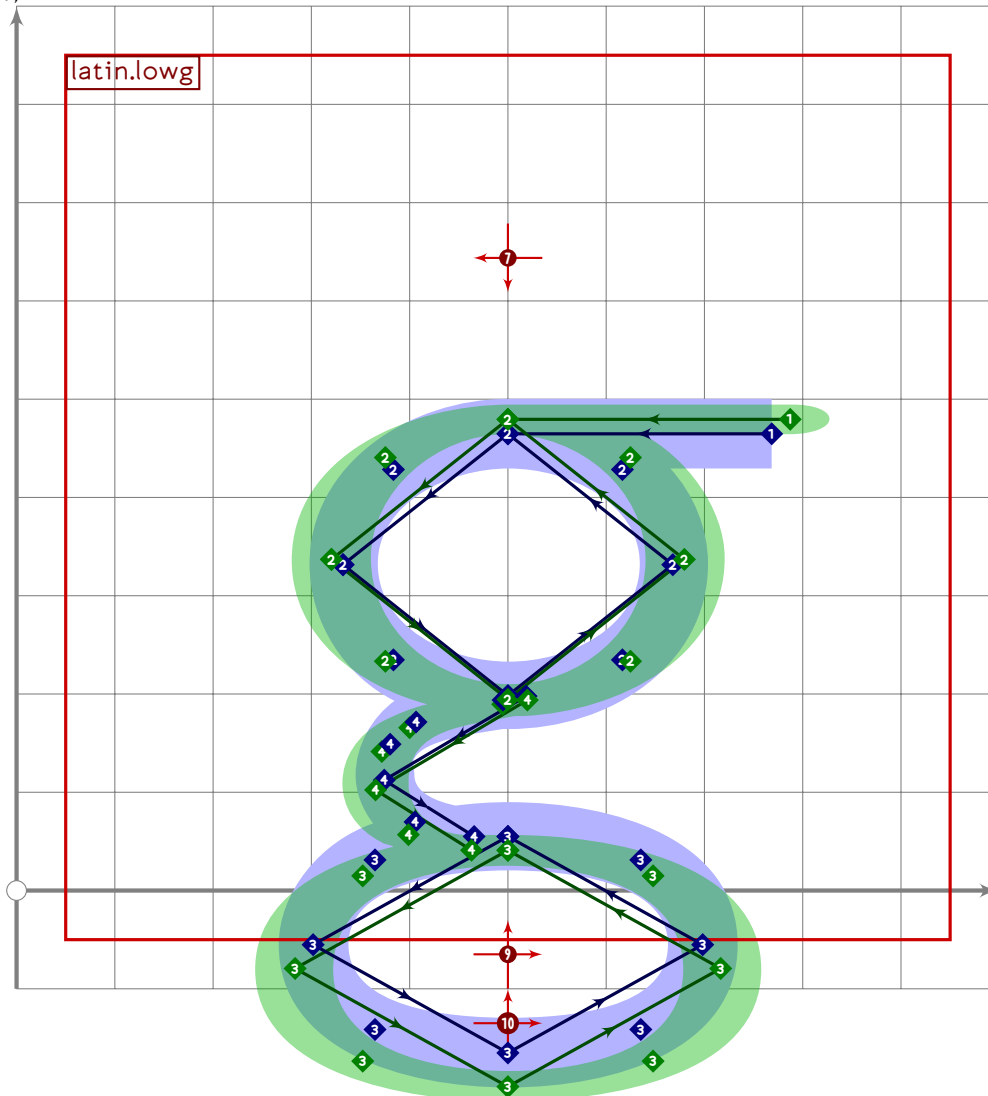
```



```

1204
1205 tsu_accent.shift_anchors(ypart olda>vmetric(0.52))
1206 (((0,0) transformed tsu_xf.cap_upper_accent)-
1207 ((0,0) transformed accent_default[anc_upper])+(70,0));
1208 expand_pbox;
1209 enddef;

```



```

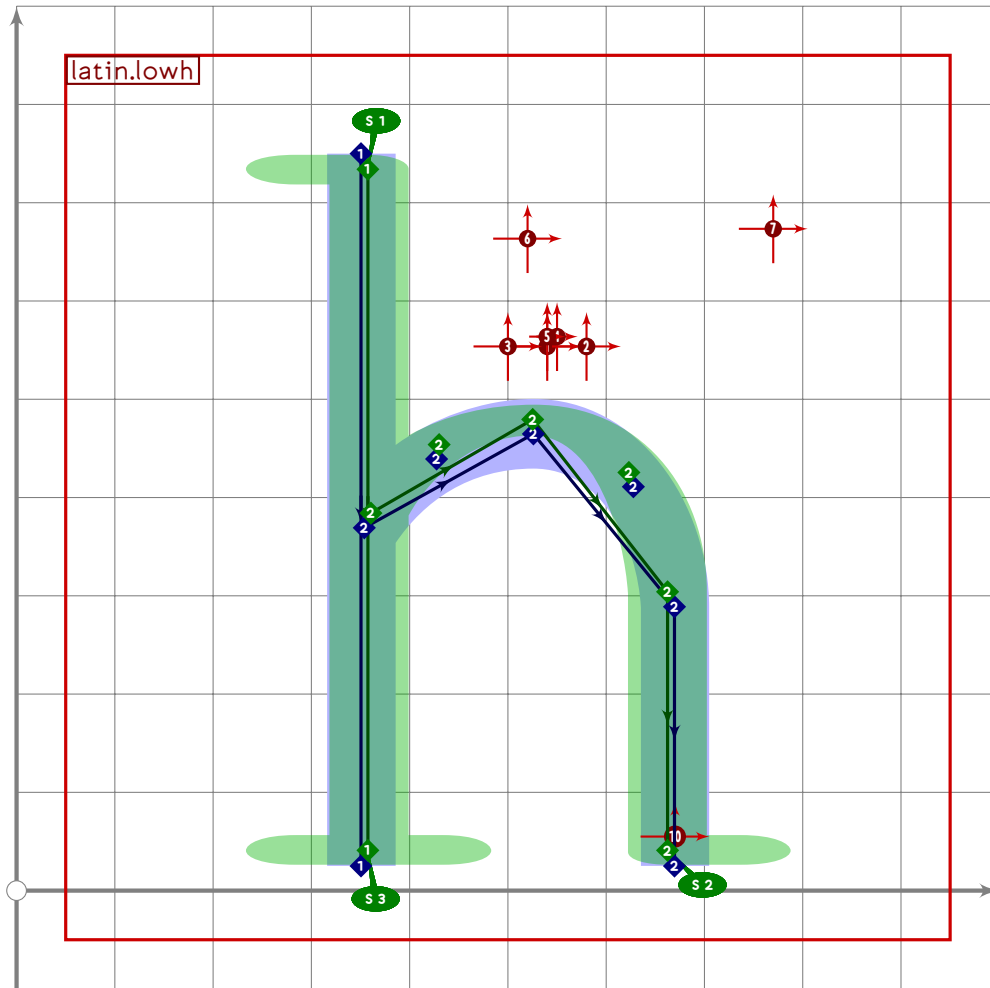
1210
1211 vardef latin.lowg =
1212   push_pbox_toexpand("latin.lowg");
1213   x2=x4=x7=x9=500;
1214   x1=1.6[x2,x5];
1215   x5-x2=x2-x3;
1216   x5-x3=1.26*(y2-y4);
1217   x6=0.25[x3,x2];
1218   x10-x7=x7-x8;
1219   x10-x8=1.8*(y7-y9);
1220
1221   y1=y2=latin_wide_xheight_h;

```

```

1222 y3=y5=0.5[y4,y2];
1223 y4=0.35[y7,y2];
1224 y6=0.4[y7,y4];
1225 y7=latin_wide_low_h;
1226 y8=y10=0.5[y9,y7];
1227 y9=latin_wide_desc_r;
1228
1229 push_stroke(z1-z2,(1.6,1.6)-(1.6,1.6));
1230
1231 push_stroke(z3..z4..z5..z2..cycle,
1232   (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-cycle);
1233
1234 push_stroke(z7..z8..z9..z10..cycle,
1235   (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-cycle);
1236
1237 push_stroke((point 1.2 of get_stroke(-1))
1238   {-direction 1.2 of get_stroke(-1)}..z6..
1239   (point 0.05 of get_stroke(0)) {-direction 0.05 of get_stroke(0)},
1240   (1.6,1.6)-(1.6,1.6)-(1.6,1.6));
1241 replace_stroke(0)(subpath (0.13,1.87) of oldp);
1242 set_bosize(0,85);
1243
1244 push_anchor(anc_caron_comma,
1245   identity rotated 180 shifted (0,90)
1246   transformed accent_default[anc_upper]);
1247 push_anchor(anc_lower,
1248   accent_default[anc_lower] shifted (0,53));
1249 push_anchor(anc_lower_connect,
1250   accent_default[anc_lower_connect] shifted (0,-190));
1251 expand_pbox;
1252 endif;

```



```

1253
1254 vardef latin.lowh =
1255   push_pbox__toexpand("latin.lowh");
1256   (x1+x5)/2=510;
1257   (x5-x1)=(y1-y2)*0.44;
1258   x2=x1=x3;
1259   x4=0.55[x3,x5];
1260   x6=x5;
1261
1262   y1=latin_wide_high_v;
1263   y2=y6=latin_wide_low_v;
1264   y3=0.77[y2,y4];
1265   y4=latin_wide_xheight_h;
1266   y5=0.60[y2,y4];
1267
1268   push_stroke(z1-z2,(1.6,1.6)-(1.6,1.6));
1269   if not do_italic_hook:
1270     set_boserif(0,0,1);
1271     set_boserif(0,1,3);
1272   fi;
1273

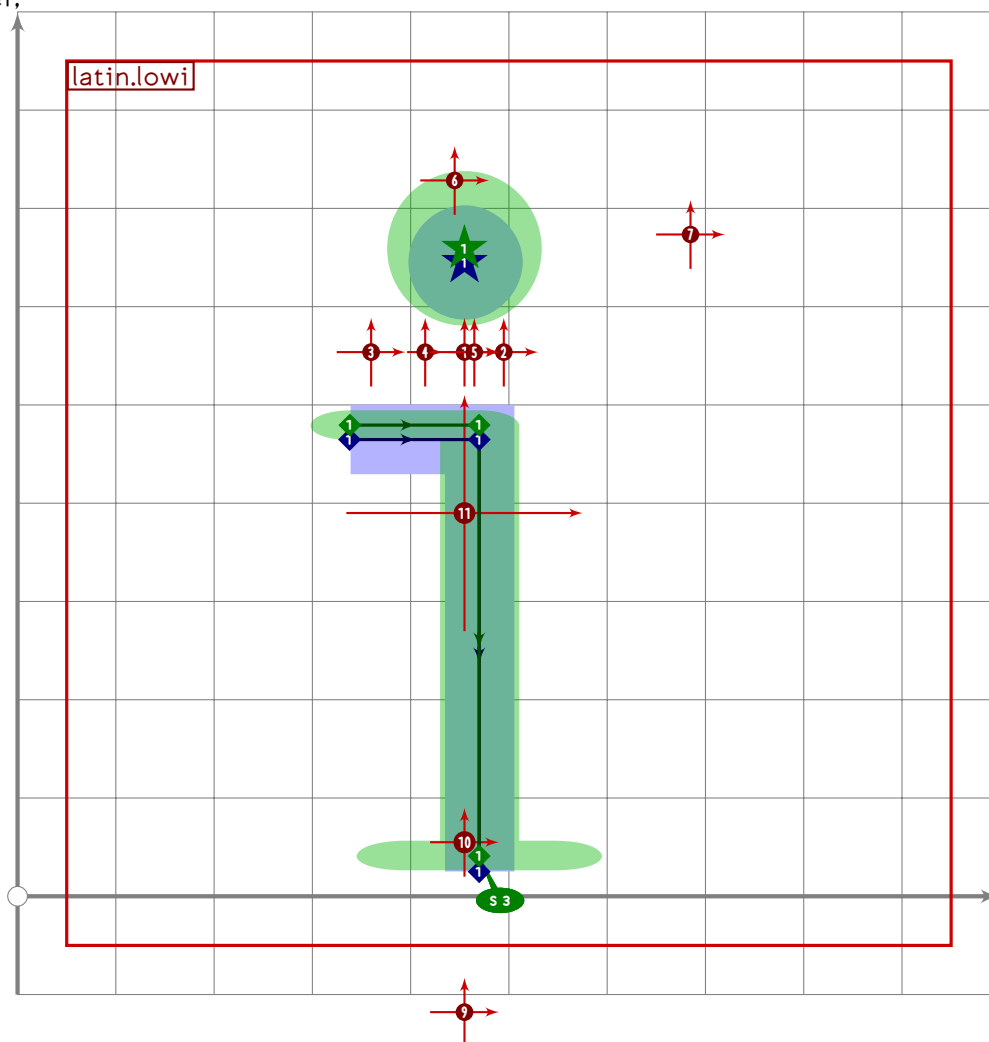
```

U+FF49
tsuku.uniFF49

```

1274 push_stroke(z3..z4{right}.z5{dir 273})-z6,
1275   (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
1276 replace_stroke(0)(subpath (0.03,3) of oldp);
1277 set_boserif(0,3;if do_italic_hook: 11 else: 2 fi);
1278
1279 tsu_accent.shift_anchors(ypart olda>vmetric(0.52))((40,0));
1280 push_anchor(anc_wide,identity xscaled 0.8
1281   transformed accent_default[anc_wide] shifted (50,10));
1282 push_anchor(anc_tilde,identity xscaled 0.7
1283   transformed accent_default[anc_tilde] shifted (40,10));
1284 push_anchor(anc_ring,accent_default[anc_ring] shifted (20,10));
1285 push_anchor(anc_lower_connect,
1286   accent_default[anc_lower_connect] shifted (170,0));
1287 expand_pbox;
1288 endif;

```



LATI

```

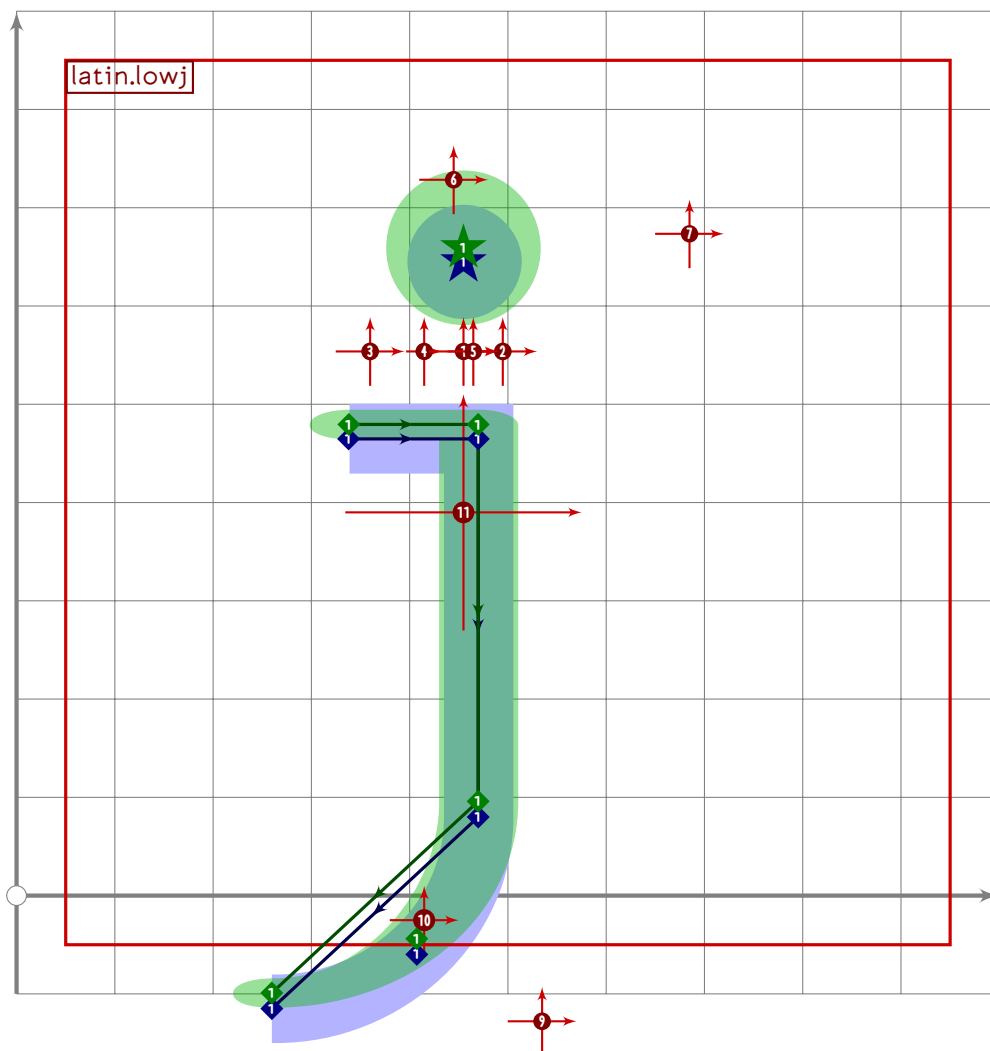
1289
1290 vardef latin.lowi =
1291   push_pbox_toexpand("latin.lowi");
1292   x2=x3;
1293   0.85[x1,x2]=450;

```

```

1294 (x2-x1)=0.3(y2-y3);
1295 x4=x2-15;
1296
1297 y1=y2=latin_wide_xheight_h;
1298 y3=latin_wide_low_v;
1299 y4=0.5[y2,latin_wide_high_v]+mbrush_width;
1300
1301 push_stroke(z1-z2-z3,(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
1302 set_botip(0,1,1);
1303 set_boserif(0,2,3);
1304
1305 push_lcblob(fullcircle rotated 45 scaled (mbrush_width*2.7+15)
1306   shifted (z4 transformed tsu_rescale_xform)
1307   transformed inverse tsu_rescale_xform);
1308
1309 push_anchor(anc_wide,
1310   identity xscaled 0.7 transformed accent_default[anc_wide]);
1311 tsu_accent.shift_anchors(true)((x4-500,0));
1312 tsu_accent.shift_anchors(ai=anc_acute)((-55,0));
1313 tsu_accent.shift_anchors(ai=anc_wide)((-40,0));
1314 tsu_accent.shift_anchors(ai=anc_tilde)((10,0));
1315 tsu_accent.shift_anchors(ai=anc_ring)((-10,55));
1316 expand_pbox;
1317 endif;

```



```

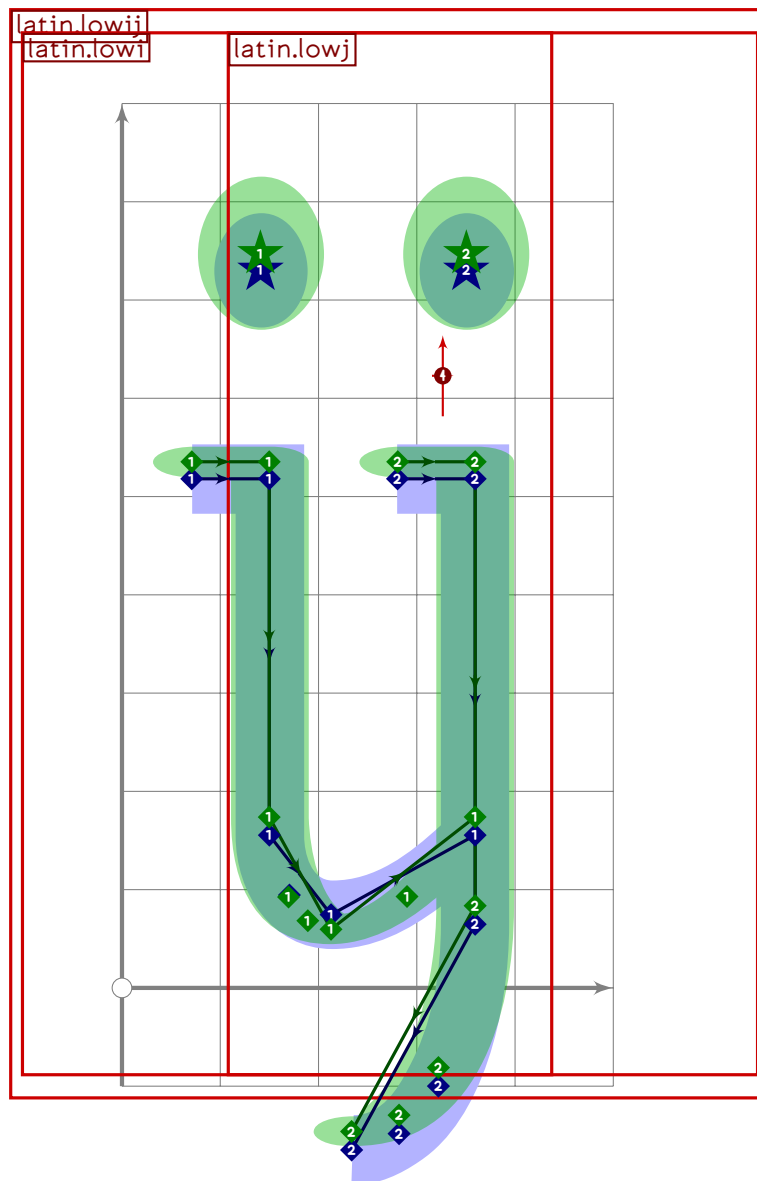
1318
1319 vardef latin.lowj =
1320   push_pbox__toexpand("latin.lowj");
1321   x2=x3;
1322   0.85[x1,x2]=450;
1323   (x2-x1)=0.3(y2-y3);
1324   x5=x2-15;
1325
1326   y1=y2=latin_wide_xheight_h;
1327   y3=latin_wide_low_v;
1328   y5=0.5[y2,latin_wide_high_v]+mbrush_width;
1329
1330   z4=z3+(-210,-140);
1331
1332   push_stroke((z1-z2-(z3+(0,55)))..{curl 0.8}z4,
1333     (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
1334   set_botip(0,1);
1335
1336   push_lcblob(fullcircle scaled (mbrush_width*2.7+15)

```

```

1337 shifted (z5 transformed tsu_rescale_xform)
1338 transformed inverse tsu_rescale_xform);
1339
1340 push_anchor(anc_wide,
1341 identity xscaled 0.7 transformed accent_default[anc_wide]);
1342 tsu_accent.shift_anchors(true)((x5-500,0));
1343 tsu_accent.shift_anchors(ai=anc_acute)((-55,0));
1344 tsu_accent.shift_anchors(ai=anc_wide)((-40,0));
1345 tsu_accent.shift_anchors(ai=anc_tilde)((10,0));
1346 tsu_accent.shift_anchors(ai=anc_ring)((-10,55));
1347 tsu_accent.shift_anchors(ai=anc_lower)((80,-10));
1348 tsu_accent.shift_anchors(ai=anc_lower_connect)((-40,-80));
1349 expand_pbox;
1350 endif;

```



```

1351
1352 vardef latin.lowij =

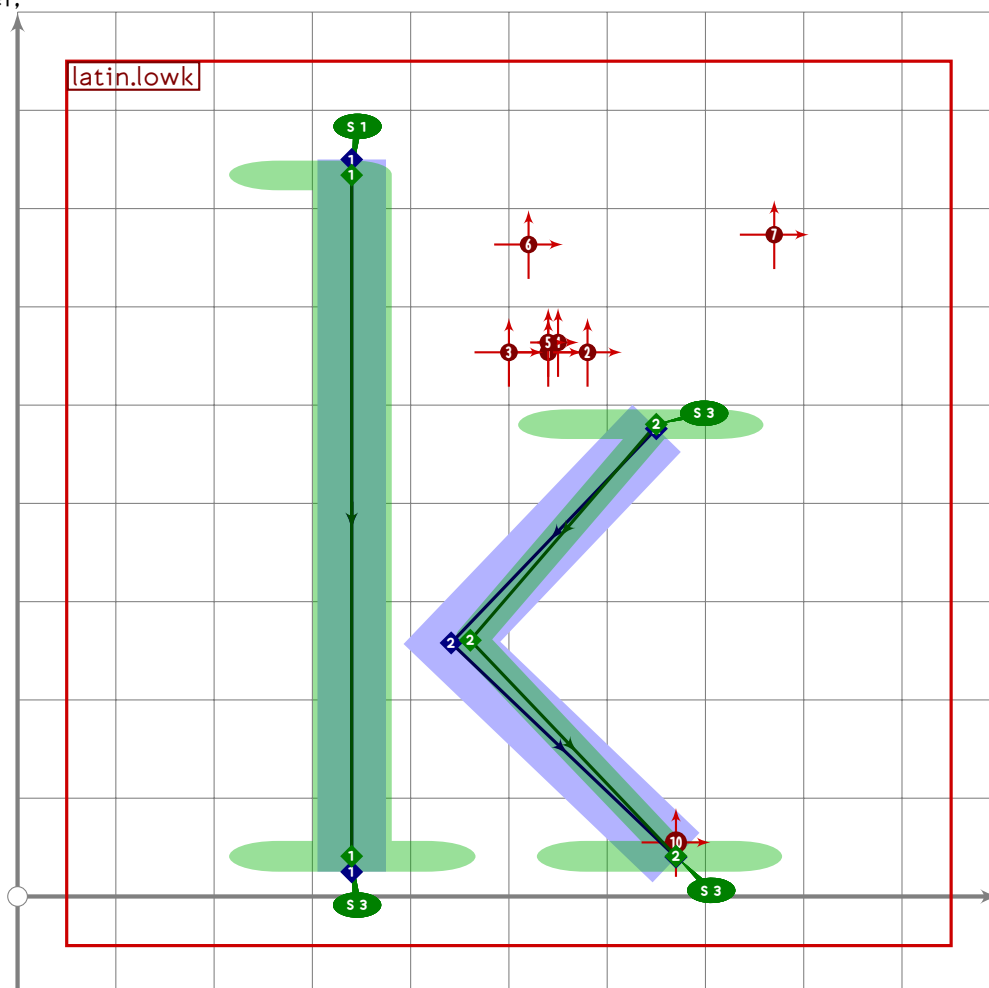
```

U+FF4B
tsuku.uniFF4B

```

1353 push_pbox_toexpand("latin.lowij");
1354 tsu_xform(identity shifted (-200,0))(latin.lowi);
1355 numeric x[],y[];
1356 tsu_xform(identity shifted (150,0))(latin.lowj);
1357 replace_lcblob(-1)(get_lcblob(0) shifted (-350,0));
1358 numeric x[],y[];
1359 z1=point 1.7 of get_stroke(-1);
1360 x2=0.3[x1,x3];
1361 y2=vmetric(0.05);
1362 z3=point 1.8 of get_stroke(0);
1363 replace_stroke(-1)((subpath (0,1) of oldp)–
1364   z1{dir 273}..z2{right}..{curl 0.3}z3);
1365 replace_strokeq(-1)((subpath (0,1) of oldq)–(1.6,1.6)–(1.6,1.6)–(1,1));
1366 set_boserif(-1,2,whatever);
1367 expand_pbox;
1368 enddef;

```



```

1369
1370 vardef latin.lowk =
1371   push_pbox_toexpand("latin.lowk");
1372   z1=(340,latin_wide_high_v);
1373   z2=(340,latin_wide_low_v);

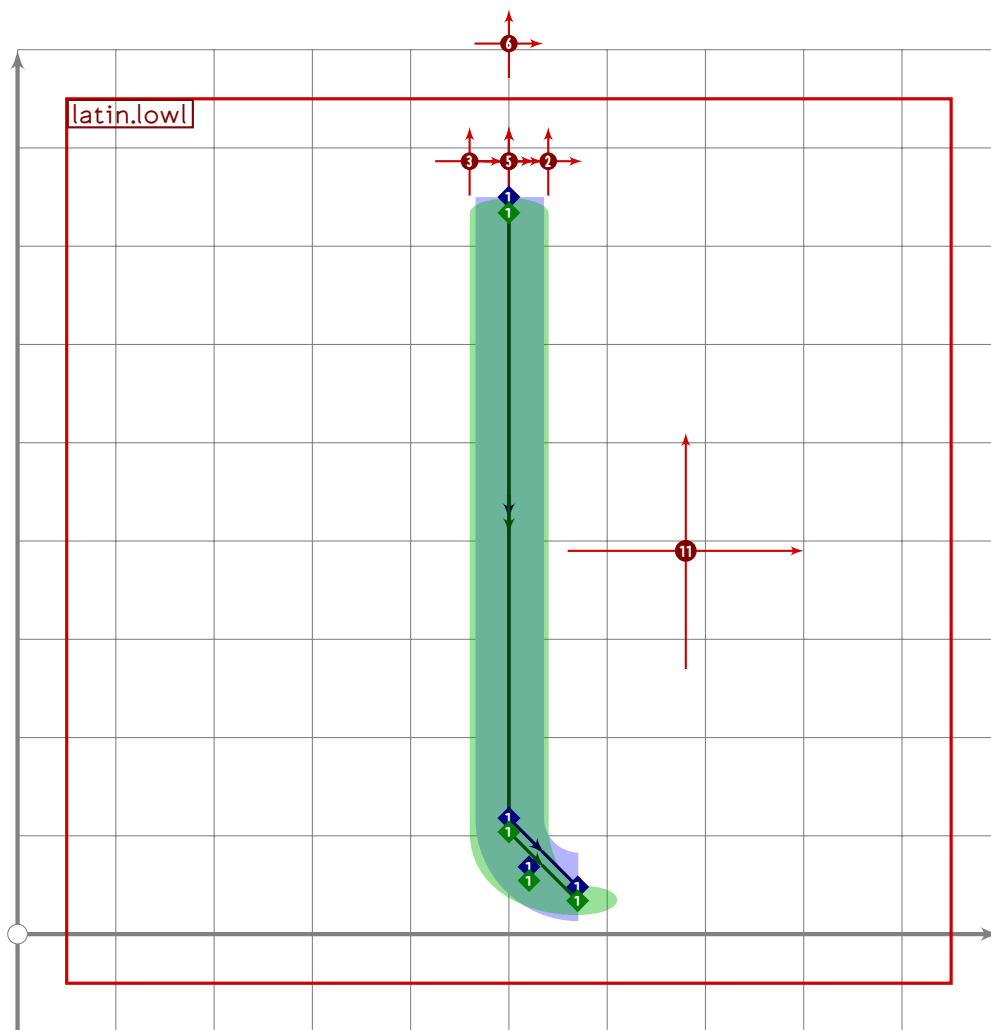
```

LATI


```

1374 z3=(650,(latin_wide_xheight_v+latin_wide_xheight_h)/2);
1375 x4=340+mbrush_width*if sharp_corners: 2.7 else: 2.3 fi;
1376 y4=(y3+y5)/2;
1377 z5=(670,0.5[latin_wide_low_h,latin_wide_low_v]);
1378
1379 push_stroke(z1-z2,(1.6,1.6)-(1.6,1.6));
1380 if not do_italic_hook:
1381     set_boserif(0,0,1);
1382     set_boserif(0,1,3);
1383 fi;
1384
1385 push_stroke(z3-z4-z5,(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
1386 set_botip(0,1,1);
1387 set_boserif(0,0,if do_italic_hook: 1 else: 3 fi);
1388 if not do_italic_hook: set_boserif(0,2,3); fi;
1389 set_bobrush(0,bralternate);
1390
1391 tsu_accent.shift_anchors(ypart olda>vmetric(0.52))((40,0));
1392 push_anchor(anc_wide,identity xscaled 0.8
1393     transformed accent_default[anc_wide] shifted (50,10));
1394 push_anchor(anc_tilde,identity xscaled 0.7
1395     transformed accent_default[anc_tilde] shifted (40,10));
1396 push_anchor(anc_ring,accent_default[anc_ring] shifted (20,-10));
1397 push_anchor(anc_lower_connect,
1398     accent_default[anc_lower_connect] shifted (170,0));
1399 expand_pbox;
1400 enddef;

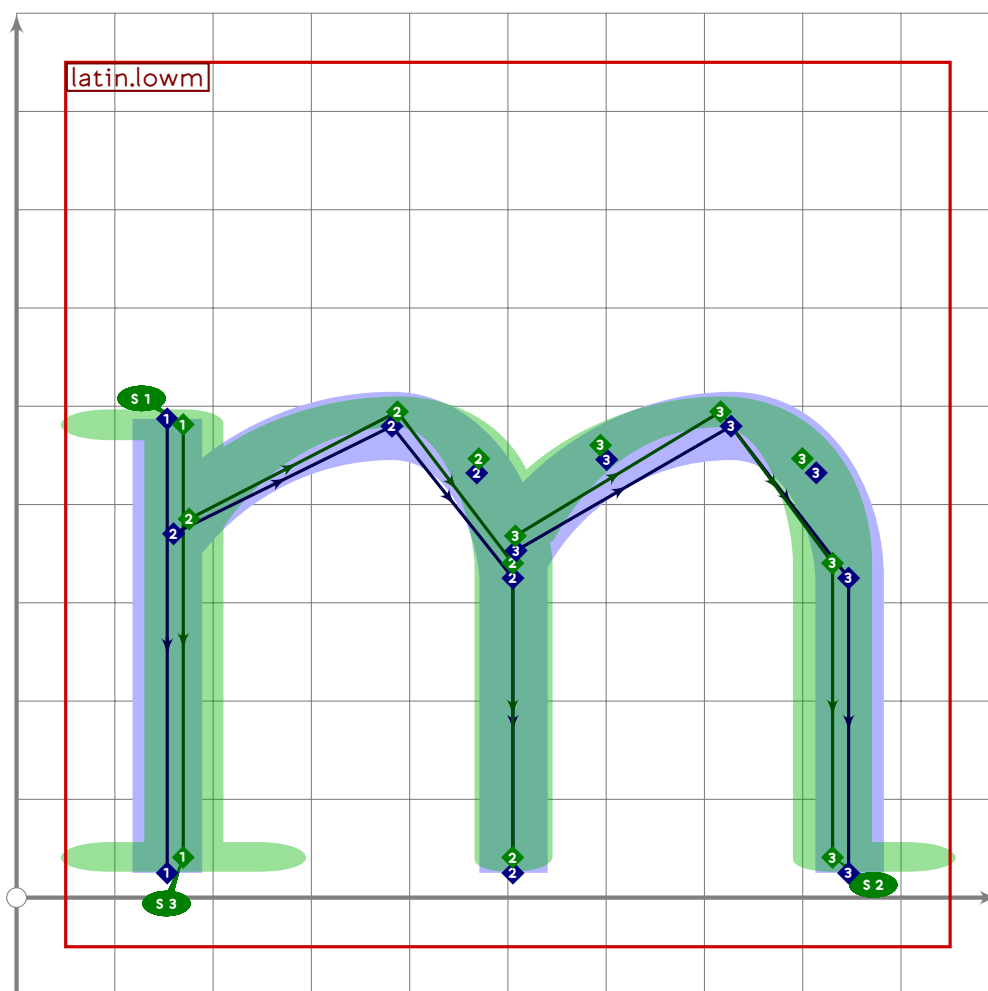
```



```

1401
1402 vardef latin.lowl =
1403   push_pbox_toexpand("latin.lowl");
1404   x1=x2=500;
1405   x3=570;
1406
1407   y1=latin_wide_high_v;
1408   y2=y3+(x3-x2);
1409   y3=latin_wide_low_r;
1410
1411   push_stroke(z1-z2{down}..{right}z3,(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
1412
1413   tsu_accent.shift_anchors((ypart olda>vmetric(0.52))
1414                           and not (ai=anc_caron_comma))
1415     (((0,0) transformed tsu_xf.cap_upper_accent)-
1416     ((0,0) transformed accent_default[anc_upper]));
1417   tsu_accent.shift_anchors(ai=anc_centre)((180,0));
1418   expand_pbox;
1419 enddef;

```



```

1420
1421 vardef latin.lowm =
1422   push_pbox_toexpand("latin.lowm");
1423   (x1+x9)/2=500;
1424   (x9-x1)*2=(y1-y2)*3;
1425   (x5-x1)=(x9-x5)*1.03;
1426   x2=x1=x3;
1427   x4=0.65[x3,x5];
1428   x6=x5;
1429   x8=0.65[x6,x9];
1430   x9=x10;
1431
1432   y1=latin_wide_xheight_v;
1433   y2=y6=y10=latin_wide_low_v;
1434   y3=0.74[y2,y4];
1435   y4=y8=latin_wide_xheight_r;
1436   y5=y9=0.66[y2,y4];
1437
1438   push_stroke(z1-z2,(1.6,1.6)-(1.6,1.6));
1439   set_boserif(0,0;if do_italic_hook: 11 else: 1 fi);
1440   if not do_italic_hook: set_boserif(0,1,3); fi;

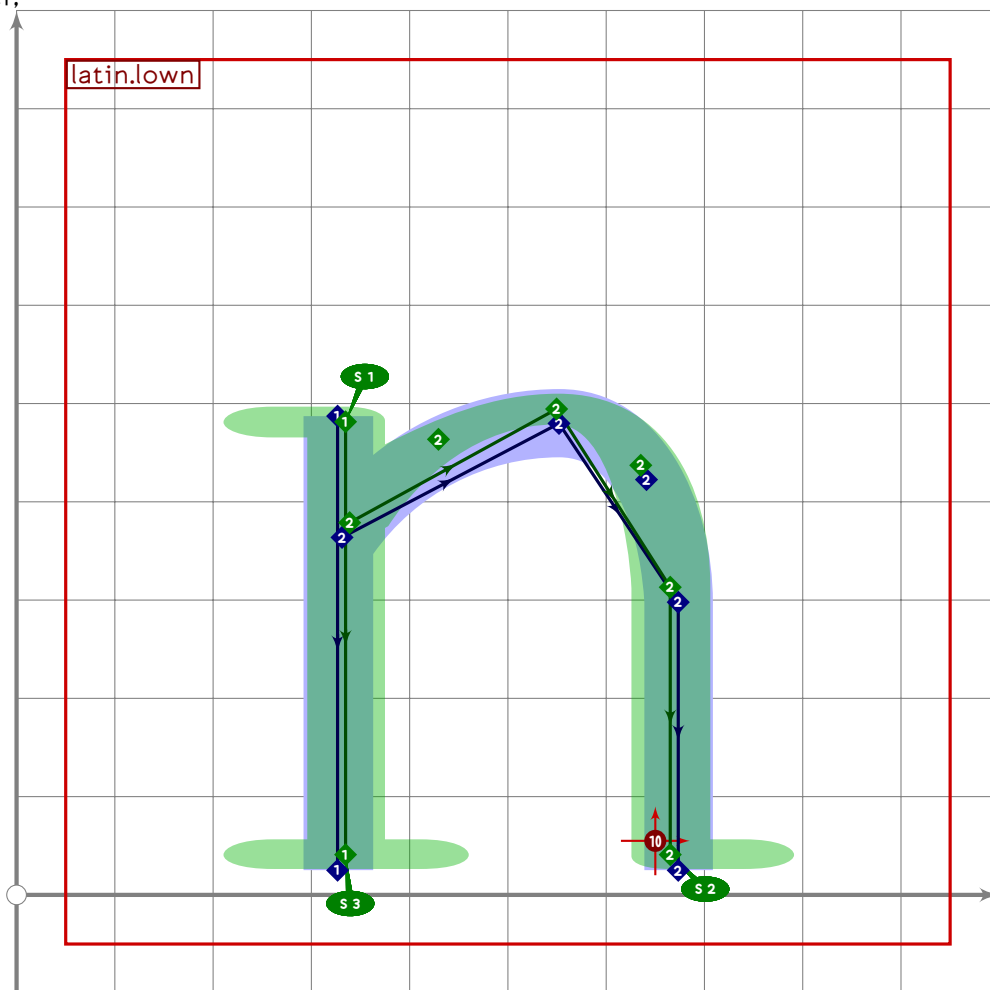
```

U+FF4E
tsuku.uniFF4E

```

1441
1442 push_stroke(z3..z4{right}..z5{dir 275}-z6,
1443   (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
1444 replace_stroke(0)(subpath (0.04,3) of oldp);
1445
1446 z7=get_stroke(0) intersectionpoint (z3-z9);
1447
1448 push_stroke(z7..z8{right}..z9{dir 271}-z10,
1449   (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
1450 replace_stroke(0)(subpath (0.04,3) of oldp);
1451 set_boserif(0,3,if do_italic_hook: 11 else: 2 fi);
1452 expand_pbox;
1453 enddef;

```



LATI

```

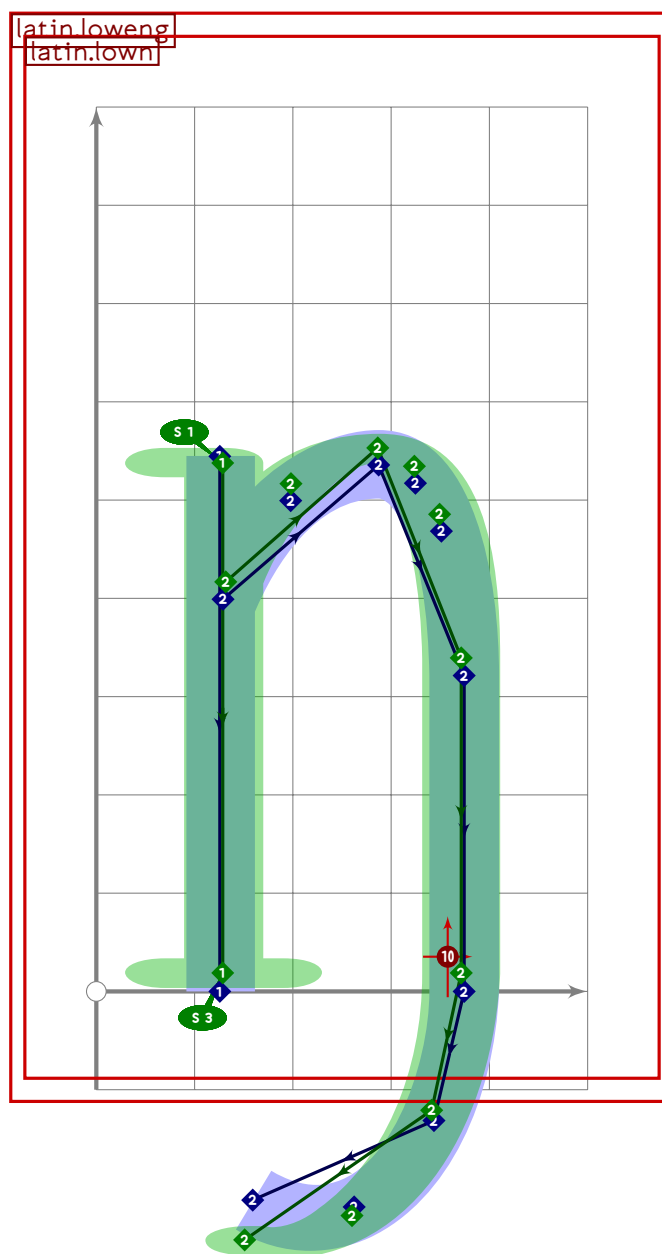
1454
1455 vardef latin.lown =
1456   push_pbox_toexpand("latin.lown");
1457   (x1+x5)/2=500;
1458   (x5-x1)=(y1-y2)*0.75;
1459   x2=x1=x3;
1460   x4=0.65[x3,x5];
1461   x6=x5;

```

```

1462
1463 y1=latin_wide_xheight_v;
1464 y2=y6=latin_wide_low_v;
1465 y3=0.73[y2,y4];
1466 y4=latin_wide_xheight_r;
1467 y5=0.60[y2,y4];
1468
1469 push_stroke(z1-z2,(1.6,1.6)-(1.6,1.6));
1470 set_boserif(0,0,if do_italic_hook: 11 else: 1 fi);
1471 if not do_italic_hook: set_boserif(0,1,3); fi;
1472
1473 push_stroke(z3..z4{right}..z5{dir 273}-z6,
1474   (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
1475 replace_strokep(0)(subpath (0.03,3) of oldp);
1476 set_boserif(0,3,if do_italic_hook: 11 else: 2 fi);
1477
1478 push_anchor(anc_lower_connect,
1479   accent_default[anc_lower_connect] shifted (150,0));
1480 expand_pbox;
1481 enddef;

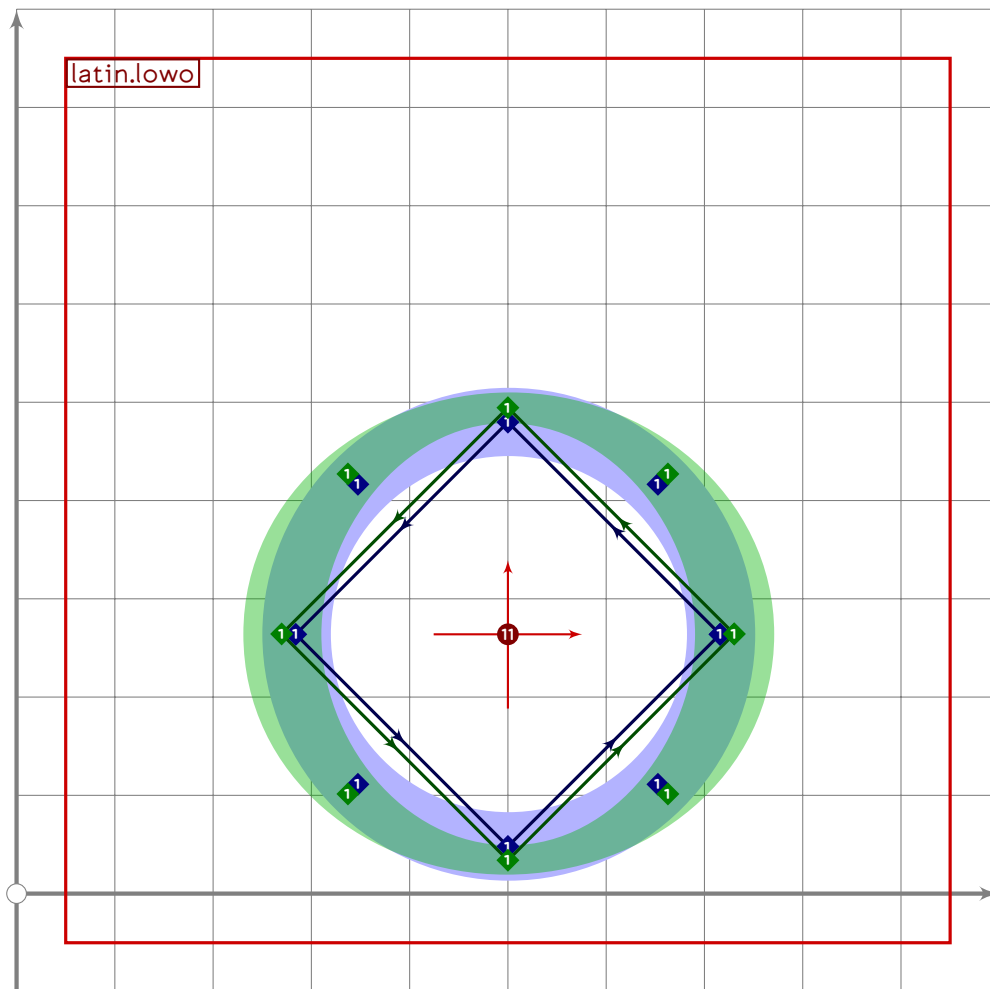
```



```

1482
1483 vardef latin.loweng =
1484   push_pbox_toexpand("latin.loweng");
1485   latin.lown;
1486   x7=x6-300;
1487   y7=latin_wide_desc_h;
1488   replace_strokep(0)(oldp{dir 266}..{curl 0.8}z7);
1489   replace_strokep(0)(insert_nodes(oldp)(3.3));
1490   replace_strokeq(0)(oldq-(1.6,1.6)-(1.6,1.6));
1491   set_boserif(0,3,whatever);
1492   expand_pbox;
1493 enddef;

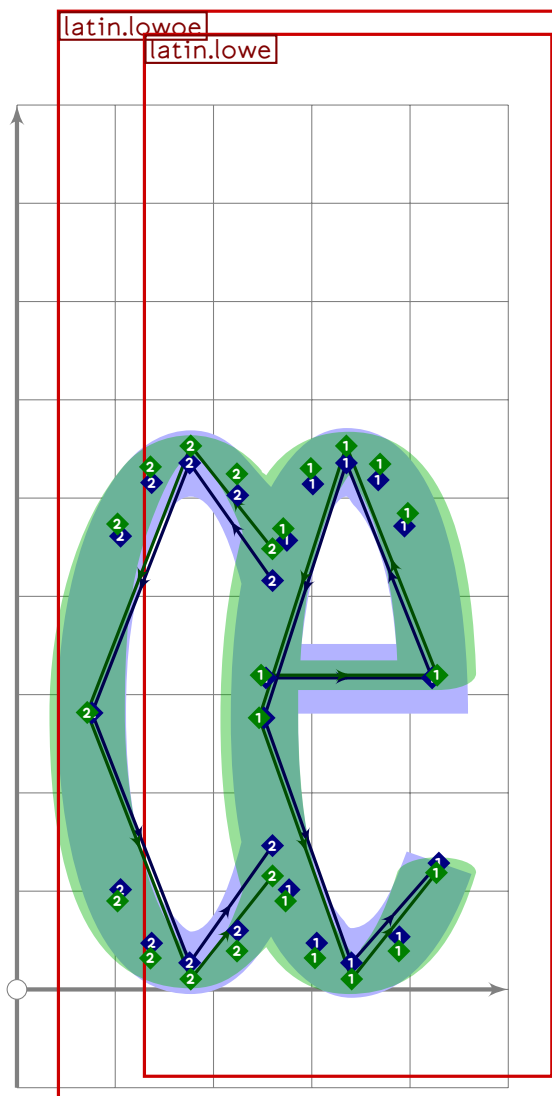
```



```

1494
1495 vardef latin.lowo =
1496   push_pbox_toexpand("latin.lowo");
1497   push_anchor(anc_centre,identity
1498     scaled ((latin_wide_xheight_r-latin_wide_low_r)/200)
1499     shifted (xpart centre_pt,(latin_wide_xheight_r+latin_wide_low_r)/2));
1500   push_stroke(((1,0)..(0,1)..(-1,0)..(0,-1)..cycle)
1501     scaled ((latin_wide_xheight_r-latin_wide_low_r)/2)
1502     shifted (xpart centre_pt,(latin_wide_xheight_r+latin_wide_low_r)/2),
1503     (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-cycle);
1504   expand_pbox;
1505 enddef;

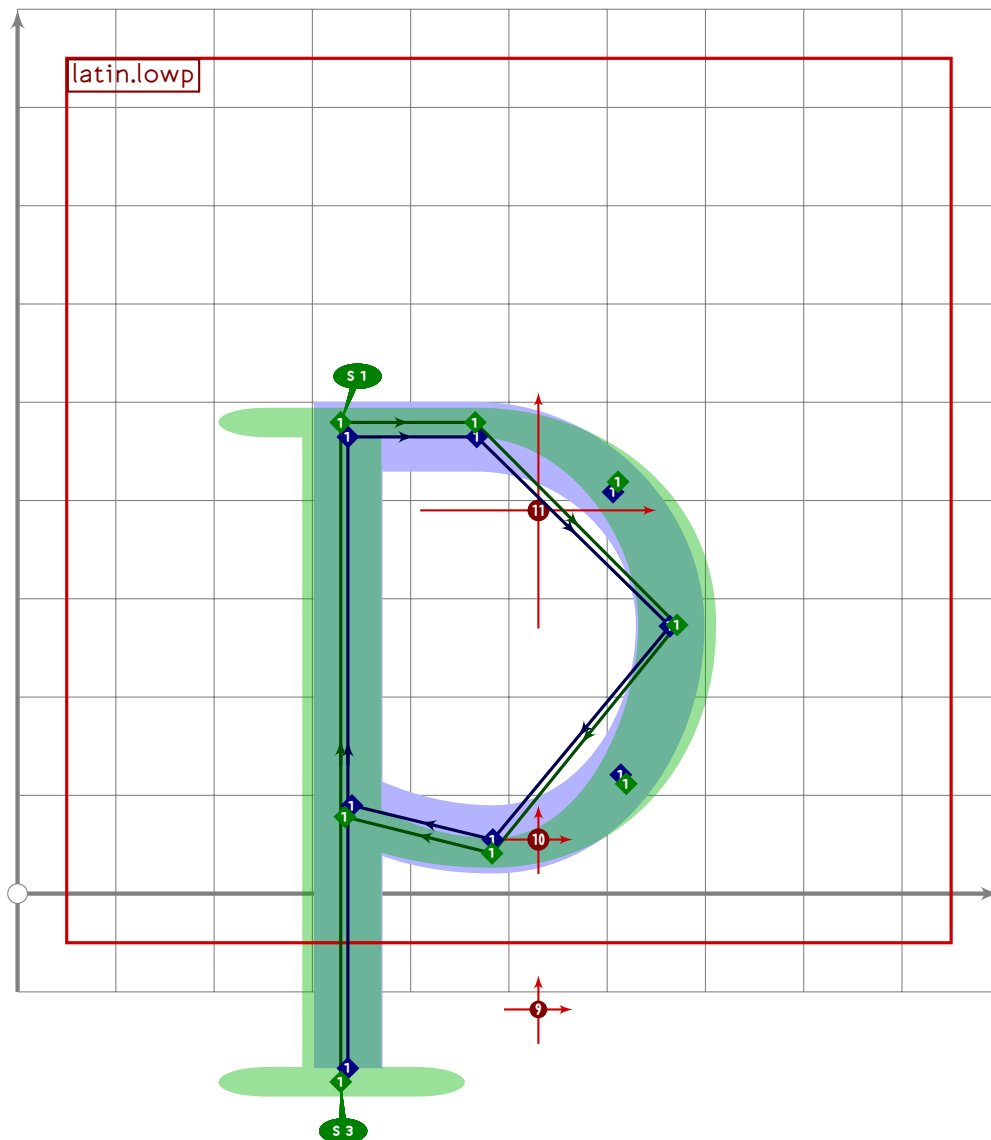
```



```

1506
1507 vardef latin.lowoe =
1508   push_pbox_toexpand("latin.lowoe");
1509   tsu_xform(identity shifted (190,0))(latin.lowe);
1510   push_stroke(((1,0)..(0,1)..(-1,0)..(0,-1)..(1,0))
1511     scaled ((latin_wide_xheight_r-latin_wide_low_r)/2)
1512     shifted (340,0.5[latin_wide_xheight_r,latin_wide_low_r]),
1513     (1.2,1.2)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.2,1.2));
1514   replace_stroke(0)(subpath (0.03+xpart (oldp intersectiontimes
1515     (subpath (2,infinity) of get_stroke(-1))),
1516     3.97-xpart ((reverse oldp) intersectiontimes
1517     reverse get_stroke(-1)))
1518     of oldp);
1519   expand_pbox;
1520 enddef;

```

```

1521
1522 vardef latin.lowp =
1523   push_pbox_toexpand("latin.lowp");
1524   (x1+x4)/2=500;
1525   (x4-x1)=(y2-y1)*0.51;
1526   x2=x1=x6;
1527   x3=0.4[x2,x4];
1528   x5=0.45[x2,x4];
1529
1530   y1=latin_wide_desc_v;
1531   y2=y3=latin_wide_xheight_h;
1532   y4=0.47[y3,y5];
1533   y5=latin_wide_low_h;
1534   y6=0.91[y3,y5];
1535
1536   push_stroke(z1-z2{right}.{right}z3.{down}z4.{left}z5..z6,
1537     (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-
1538     (1.6,1.6)-(1.6,1.6));

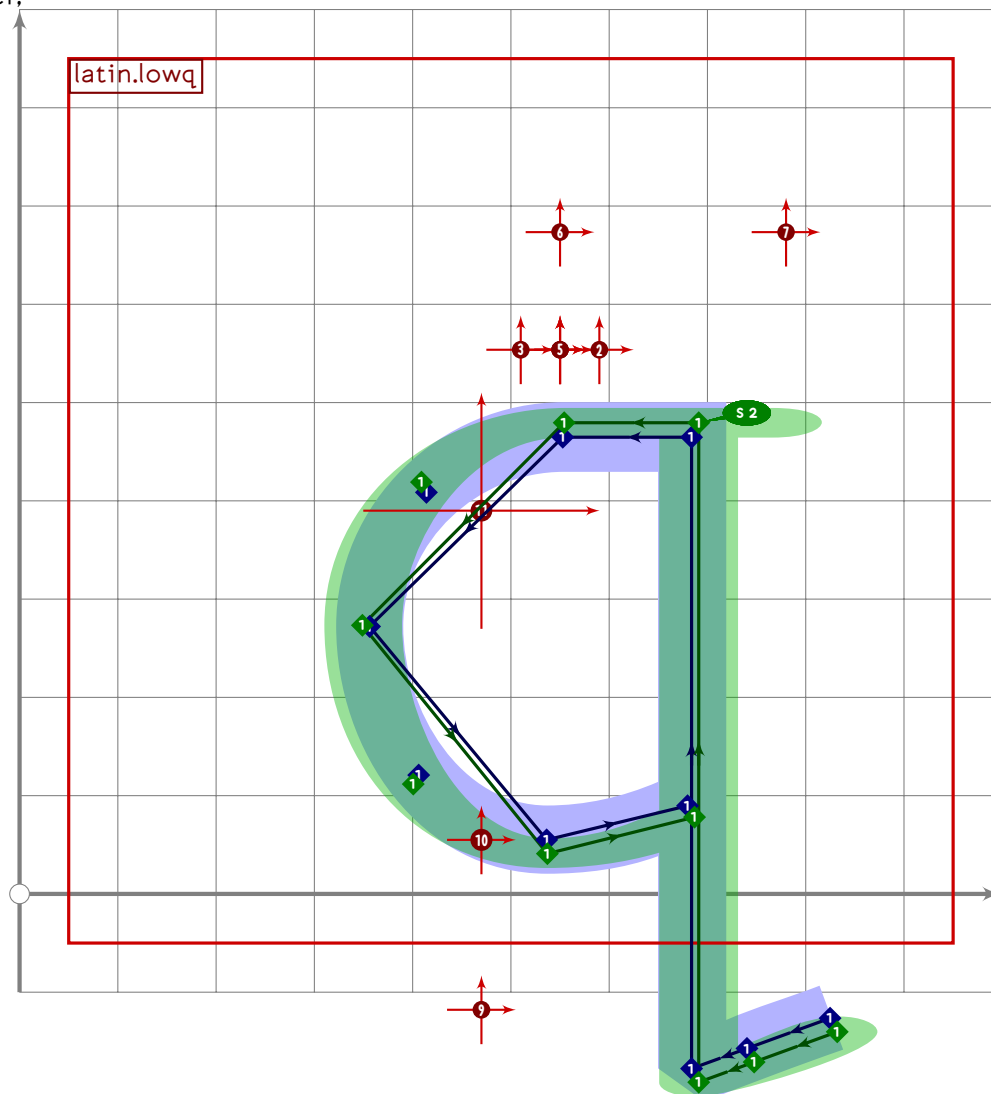
```

U+FF51
tsuku.uniFF51

```

1539 replace_strokep(0)(subpath (0,4,97) of oldp);
1540 set_botip(0,1,1);
1541 if not do_italic_hook: set_boserif(0,0,3); fi;
1542 set_boserif(0,1,1);
1543–1544
1545 tsu_accent.shift_anchors(ypart olda<vmetric(0.52))((30,0));
1546 expand_pbox;
1547 enddef;

```



```

1548
1549 vardef latin.lowq =
1550   push_pbox_toexpand("latin.lowq");
1551   (x1+x4)/2=520;
1552   (x1-x4)=(y2-y1)*0.51;
1553   x2=x1=x6;
1554   x3=0.4[x2,x4];
1555   x5=0.45[x2,x4];
1556
1557   y1=latin_wide_desc_v;

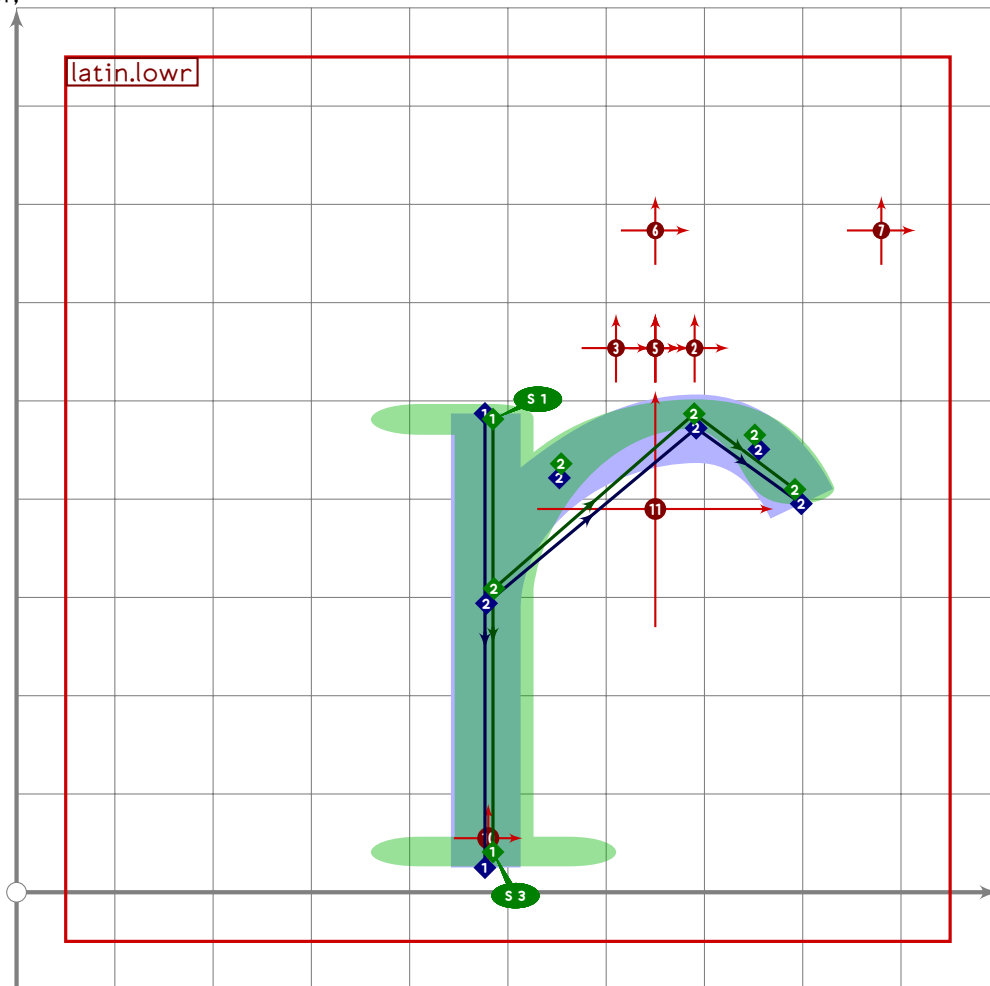
```

LATI

```

1558 y2=y3=latin_wide_xheight_h;
1559 y4=0.47[y3,y5];
1560 y5=latin_wide_low_h;
1561 y6=0.91[y3,y5];
1562
1563 z0=z1+150*(dir 20);
1564
1565 push_stroke(z0-(0.6[z0,z1])-z1-z2{left}..
1566   {left}z3..{down}z4..{right}z5..z6,
1567   (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-
1568   (1.6,1.6)-(1.6,1.6)-(1.6,1.6));
1569 replace_strokep(0)(subpath (0,6.97) of oldp);
1570 set_botip(0,2,0);
1571 set_botip(0,3,1);
1572 if not do_italic_hook: set_boserif(0,3,2); fi;
1573
1574 tsu_accent.shift_anchors(ypart olda<vmetric(0.52))((-30,0));
1575 tsu_accent.shift_anchors(ypart olda>vmetric(0.52))((50,0));
1576 expand_pbox;
1577 enddef;

```



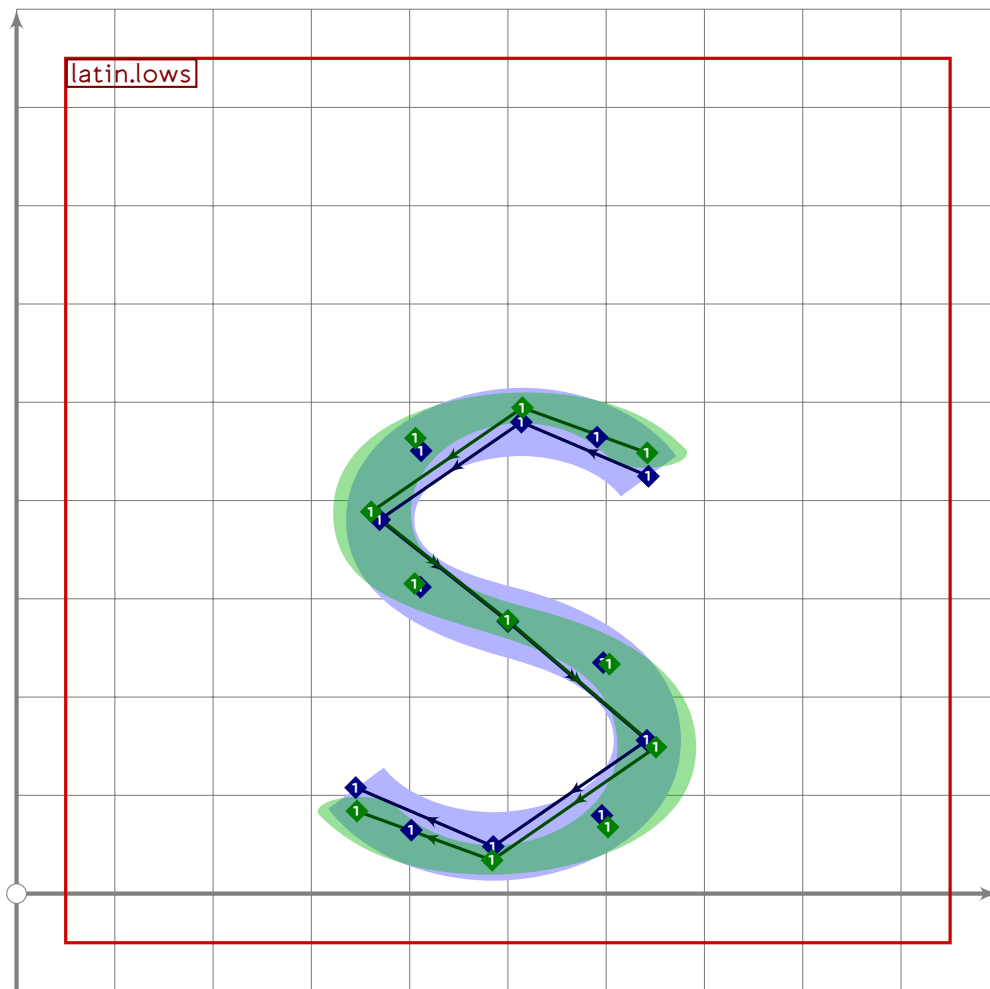
LATI

1578

```

1579 vardef latin.lowr =
1580   push_pbox_toexpand("latin.lowr");
1581   (x1+x5)/2=650;
1582   (x5-x1)=(y1-y2)*0.75;
1583   x2=x1=x3;
1584   x4=0.62[x3,x5];
1585
1586   y1=latin_wide_xheight_v;
1587   y2=latin_wide_low_v;
1588   y3=0.58[y2,y4];
1589   y4=0.5[latin_wide_xheight_h,latin_wide_xheight_r];
1590   y5=0.60[y2,y4];
1591
1592   push_stroke(z1-z2,(1.6,1.6)-(1.6,1.6));
1593   set_boserif(0,0,if do_italic_hook: 11 else: 1 fi);
1594   if not do_italic_hook: set_boserif(0,1,3); fi;
1595
1596   push_stroke(z3..z4{right}..{dir 273}z5,
1597     (1.6,1.6)-(1.6,1.6)-(1.6,1.6));
1598   replace_strokep(0)(subpath (0.03,1.6) of oldp);
1599
1600   tsu_accent.shift_anchors(y part olda>vmetric(0.05))((0.5[x3,x5]-500,0));
1601   tsu_accent.shift_anchors(ai=anc_lower_connect)((-20,0));
1602   expand_pbox;
1603 enddef;

```



```

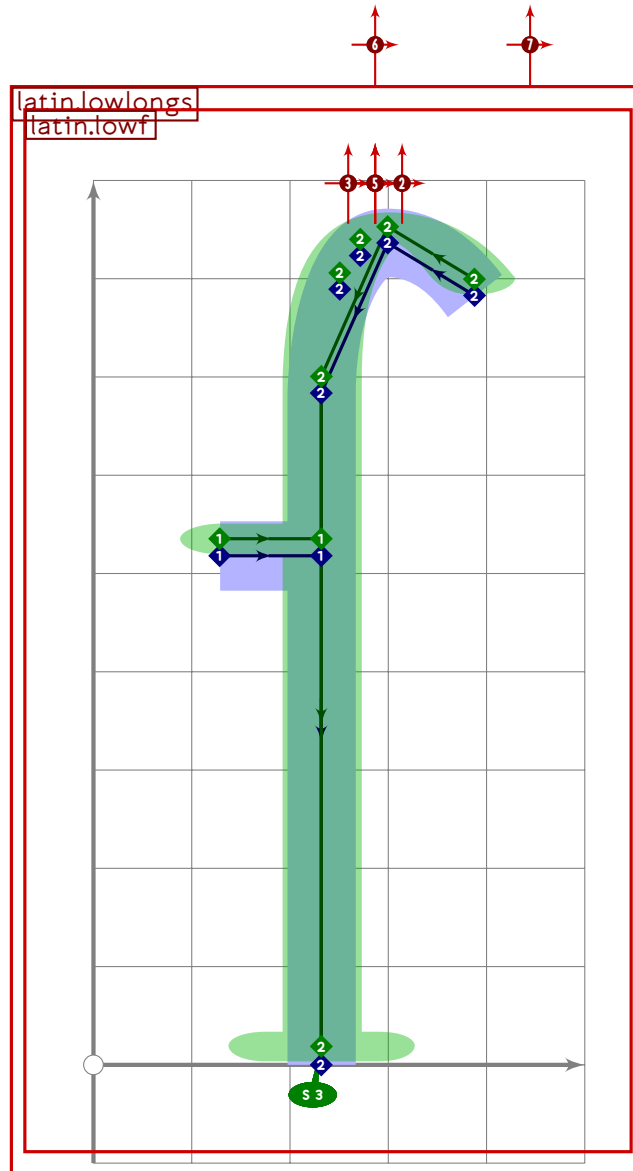
1604
1605 vardef latin.lows =
1606   push_pbox_toexpand("latin.lows");
1607   transform ta,tb;
1608   path mycurve;
1609
1610   mycurve:=(1,0)..(0,1)..(-1,0);
1611
1612   y2=latin_wide_xheight_r;
1613   y0=y3=0.77[y6,y2];
1614   y4=0.53[y6,y2];
1615   y5=y8=0.25[y6,y2];
1616   y6=latin_wide_low_r;
1617
1618   0.48[x1,x7]=0.48[x2,x6]=0.48[x3,x5]=x4=500;
1619   x5-x1=5;
1620   x5-x7=(y2-y6)*0.67;
1621
1622   (point 0 of mycurve) transformed ta=z0;
1623   (point 0.35 of mycurve) transformed ta=z1;
1624   (point 1 of mycurve) transformed ta=z2;

```

```

1625 (point 2 of mycurve) transformed ta=z3;
1626 xypart ta=0;
1627
1628 (point 0 of mycurve) transformed tb=z8;
1629 (point 0.35 of mycurve) transformed tb=z7;
1630 (point 1 of mycurve) transformed tb=z6;
1631 (point 2 of mycurve) transformed tb=z5;
1632
1633 if sharp_corners:
1634   mycurve:=subpath (0.29,2) of mycurve;
1635 else:
1636   mycurve:=subpath (0.38,2) of mycurve;
1637 fi;
1638
1639 push_stroke((mycurve transformed ta)..z4..(reverse mycurve transformed tb),
1640   (1.6,1.6)–(1.6,1.6)–(1.6,1.6)–(1.6,1.6)–(1.6,1.6)–
1641   (1.6,1.6)–(1.6,1.6));
1642 expand_pbox;
1643 endif;

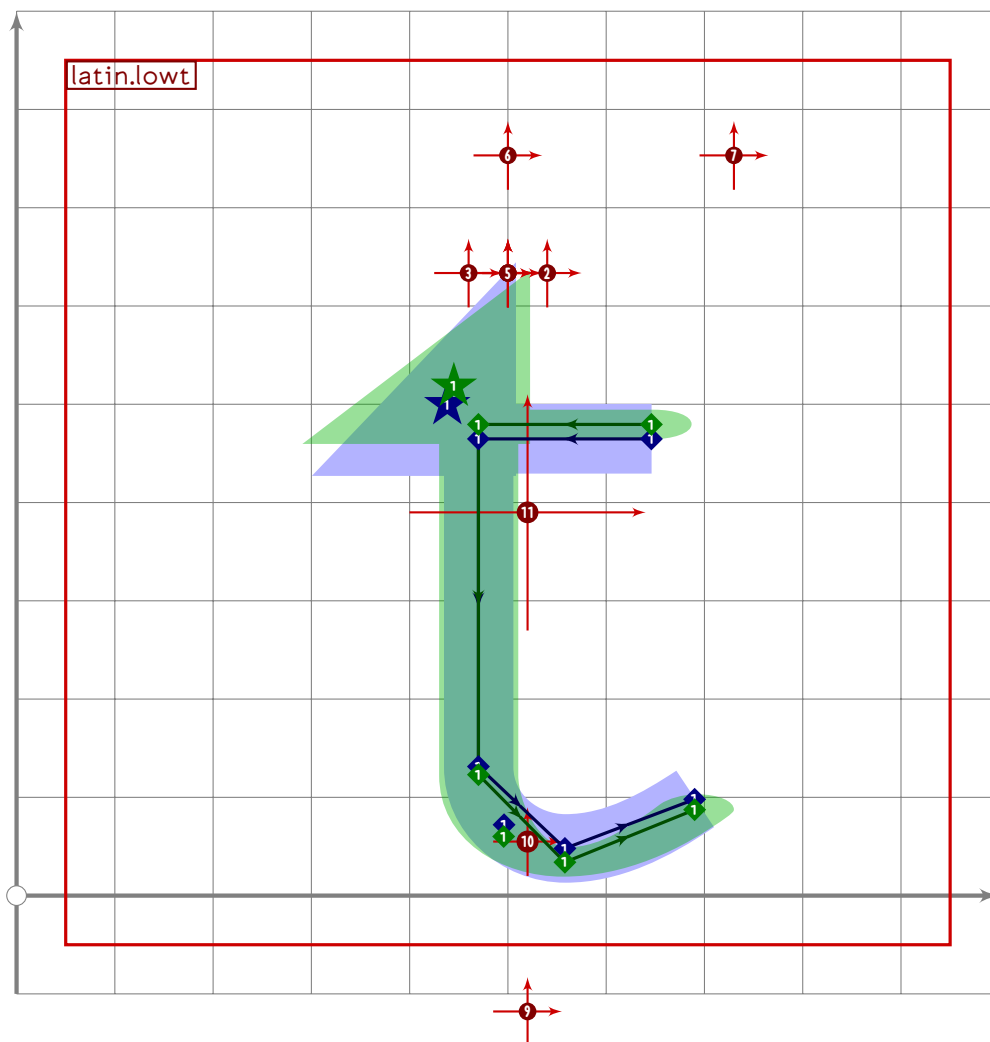
```



```

1644
1645 vardef latin.lowlongs =
1646   push_pbox_toexpand("latin.lowlongs");
1647   latin.lowf;
1648   replace_strokep(-1)(subpath (0,0.52) of oldp);
1649   expand_pbox;
1650 enddef;

```



```

1651
1652 vardef latin.lowt =
1653   push_pbox_toexpand("latin.lowt");
1654   x2=x3=x10=470;
1655   x1=0.2[x5,x2];
1656   x4=0.6[x5,x2];
1657   x5-x2=220;
1658
1659   y1=y2=y6=latin_wide_xheight_h;
1660   y3=0.2[y4,y1];
1661   y4=latin_wide_low_r;
1662   y5=0.12[y4,y1];
1663   y10=vmetric(0.83);
1664
1665   if is_proportional:
1666     z10-z6=whatever*dir 58;
1667   else:
1668     z10-z6=whatever*dir 47;
1669   fi;
1670

```

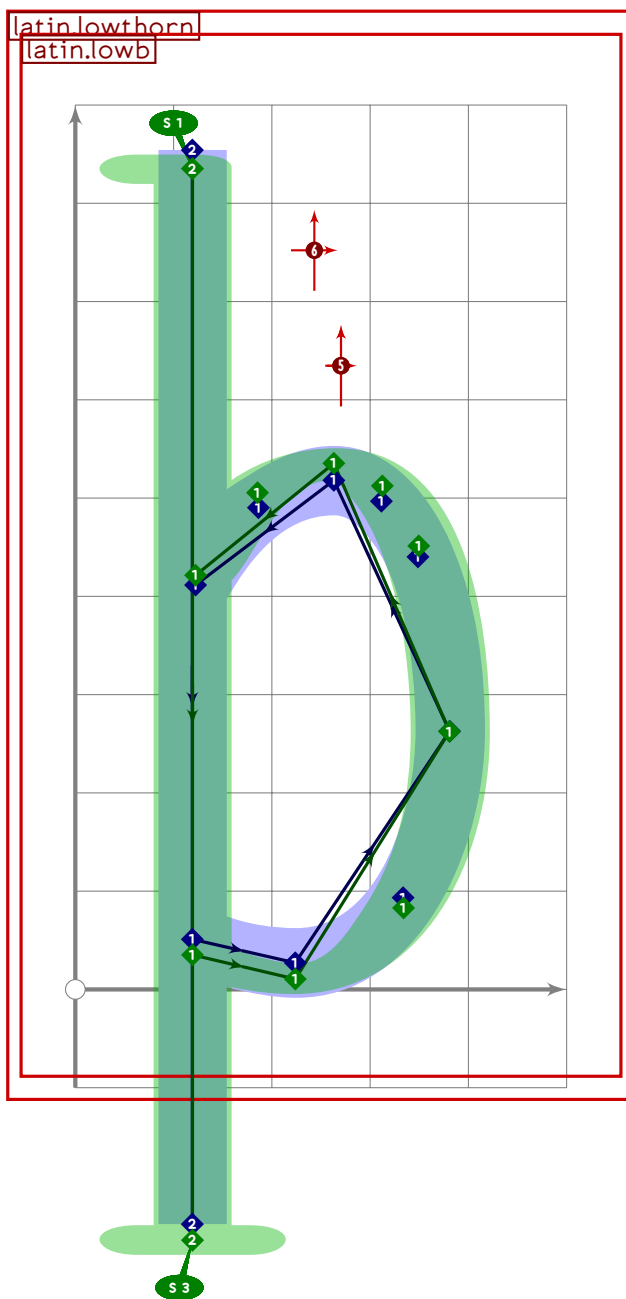


```

1671 if tsu_brush_max.brpunct*100<30:
1672     push_stroke(z1-z6-z10-z3{down}..z4{right}..{curl 0.2}z5,
1673         (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
1674     set_botip(0,1,1);
1675     set_botip(0,2,1);
1676 else:
1677     push_stroke(z1-z2-z3{down}..z4{right}..{curl 0.2}z5,
1678         (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
1679     set_botip(0,1,0);
1680 fi;
1681
1682 push_stroke(z6-z1,(0,0)-(0,0));
1683
1684 x7=x8=x2;
1685 x9=x6;
1686
1687 y7=y9=y2;
1688
1689 if is__proportional:
1690     z8-z9=whatever*dir 58;
1691 else:
1692     z8-z9=whatever*dir 47;
1693 fi;
1694
1695 if tsu_brush_max.brpunct>=0.3:
1696     begingroup
1697         save t; transform t;
1698         t:=tsu_rescale_xform;
1699         push_lcblob(((z7 transformed t)+(mbrush_width,mbrush_height))-
1700             ((z8 transformed t)+(mbrush_width,mbrush_height))-
1701             ((z9 transformed t)+(-mbrush_width,-mbrush_height))-
1702             cycle);
1703         replace_lcblob(0)(oldblob transformed inverse t);
1704     endgroup;
1705 fi;
1706
1707 tsu_accent.shift_anchors(ypart olda>vmetric(0.52))
1708     ((0,vmetric(0.12)-vmetric(0)));
1709 tsu_accent.shift_anchors(ypart olda<vmetric(0.52))
1710     ((20,0));
1711 expand_pbox;
1712 enddef;

```

U+00FE
tsuku.thorn



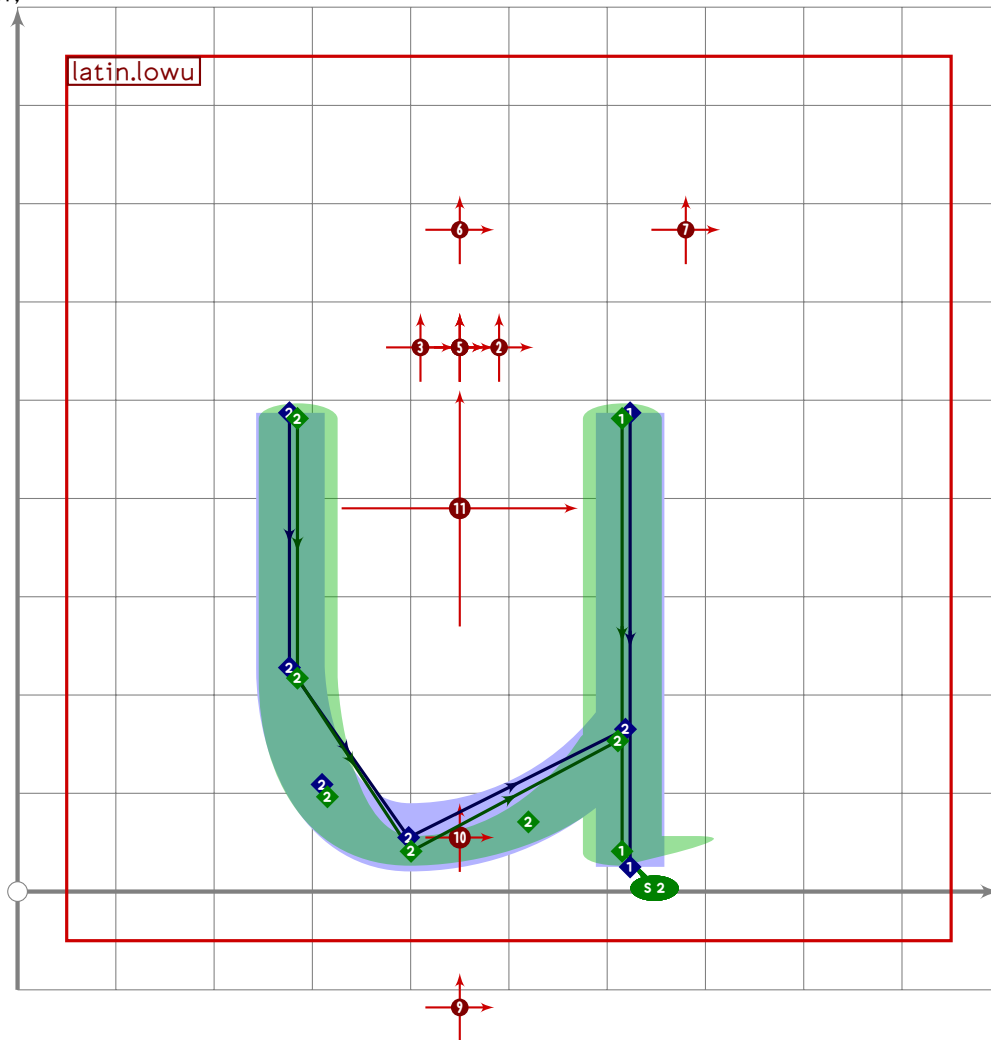
```

1713
1714 vardef latin.lowthorn =
1715   push_pbox__toexpand("latin.lowthorn");
1716   latin.lowb;
1717   set_botip(0,whatever);
1718   set_boserif(0,0,whatever);
1719   push_stroke((point 0 of get_strokep(0))--
1720     (xpart point 0 of get_strokep(0),latin_wide_desc_v,
1721     (1.6,1.6)--(1.6,1.6));
1722   set_boserif(0,0,1);
1723   set_boserif(0,1,3);
1724   replace_strokep(-1)(subpath (1,infinity) of oldp);
1725   replace_strokeq(-1)(subpath (1,infinity) of oldq);
1726   expand_pbox;

```

LATI

1727 enddef;



1728

1729 vardef latin.lowu =

1730 push_pbox_toexpand("latin.lowu");

1731 (x1+x5)/2=450;

1732 (x1-x5)=(y2-y1)*0.75;

1733 x2=x1=x3;

1734 x4=0.65[x3,x5];

1735 x6=x5;

1736

1737 y1=latin_wide_low_v;

1738 y2=y6=latin_wide_xheight_v;

1739 y3=0.73[y2,y4];

1740 y4=latin_wide_low_h;

1741 y5=0.60[y2,y4];

1742

1743 push_stroke(z2-z1,(1.6,1.6)-(1.6,1.6));

1744 set_boserif(0,if do_italic_hook: 11 else: 2 fi);

1745

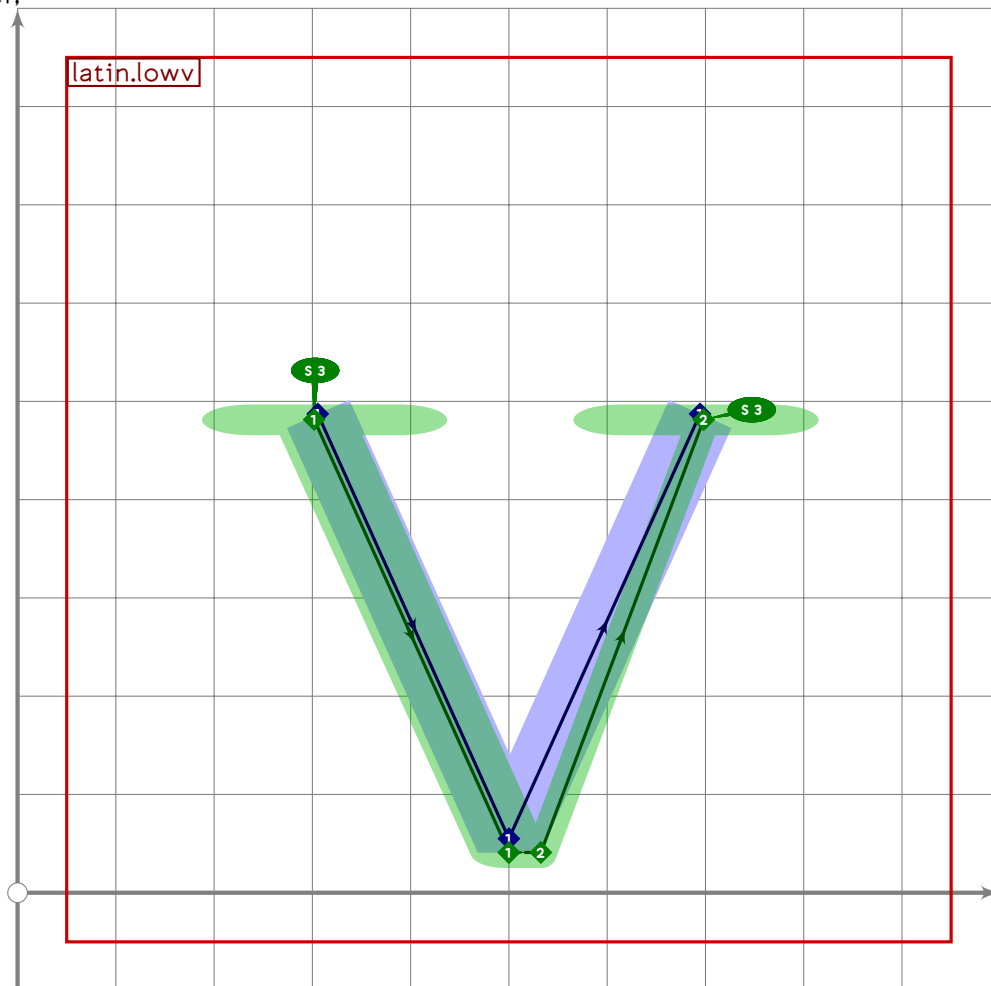
1746 push_stroke(reverse(z3..z4{left}..z5{dir 93}-z6),

U+FF56
tsuku.uniFF56

```

1747     (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
1748   replace_strokep(0)(subpath (0,2.97) of oldp);
1749   if do_italic_hook: set_boserif(0,0,11); fi;
1750
1751   tsu_accent.shift_anchors(true)((-50,0));
1752   expand_pbox;
1753 enddef;

```



```

1754
1755 vardef latin.lowv =
1756   push_pbox_toexpand("latin.lowv");
1757   (x1+x3)/2=x2=500;
1758
1759   y1=y3=latin_wide_xheight_v;
1760   y2=latin_wide_low_h;
1761
1762   (x3-x1)=(y1-y2)*0.9;
1763
1764   if do_alteration:
1765     push_stroke(z1-z2,(1.6,1.6)-(1.6,1.6));
1766     set_boserif(0,0,3);
1767

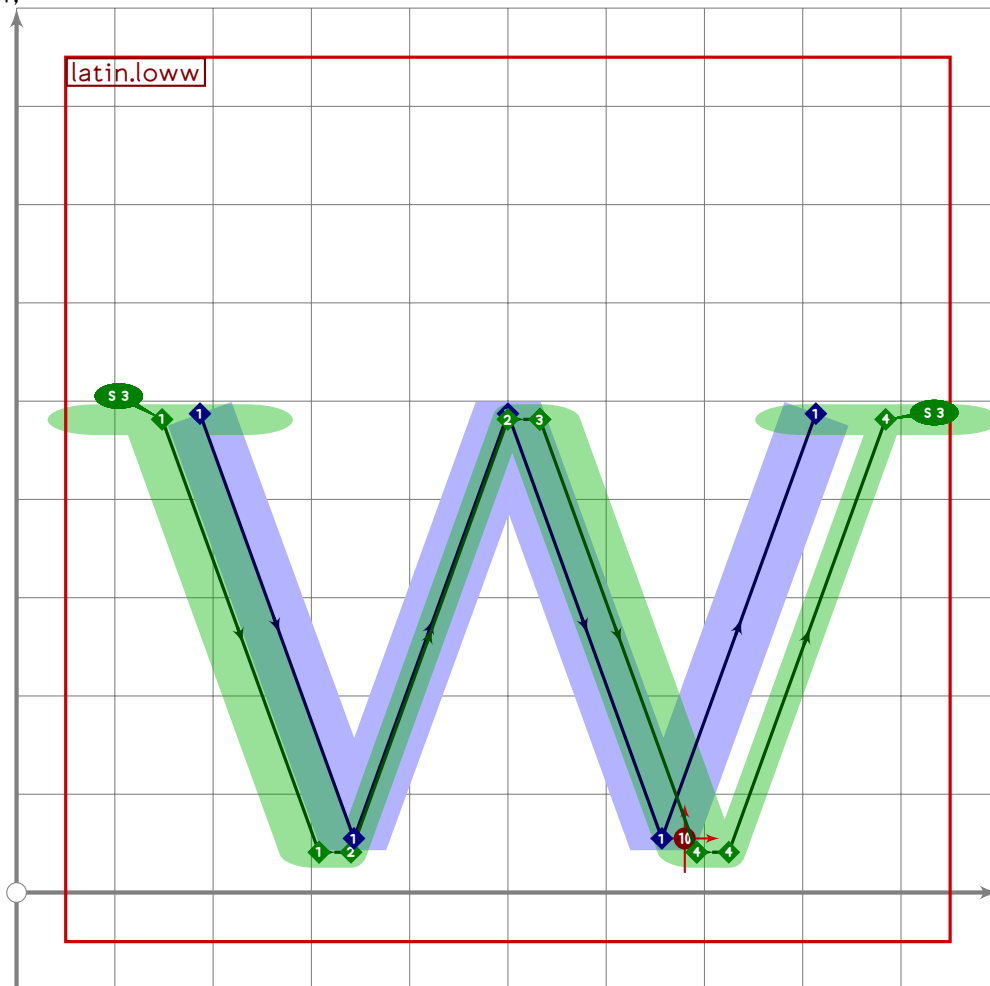
```

LATI

```

1768   push_stroke(z2-(z2+alternate_adjust*right)-z3,
1769     (1.6,1.6)-(1.6,1.6)-(1.6,1.6));
1770   set_boserif(0,2,3);
1771   set_bobrush(0,bralternate);
1772   else:
1773     push_stroke(z1-z2-z3,(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
1774     set_botip(0,1,0);
1775     set_boserif(0,0,1);
1776   fi;
1777   expand_pbox;
1778 enddef;

```



```

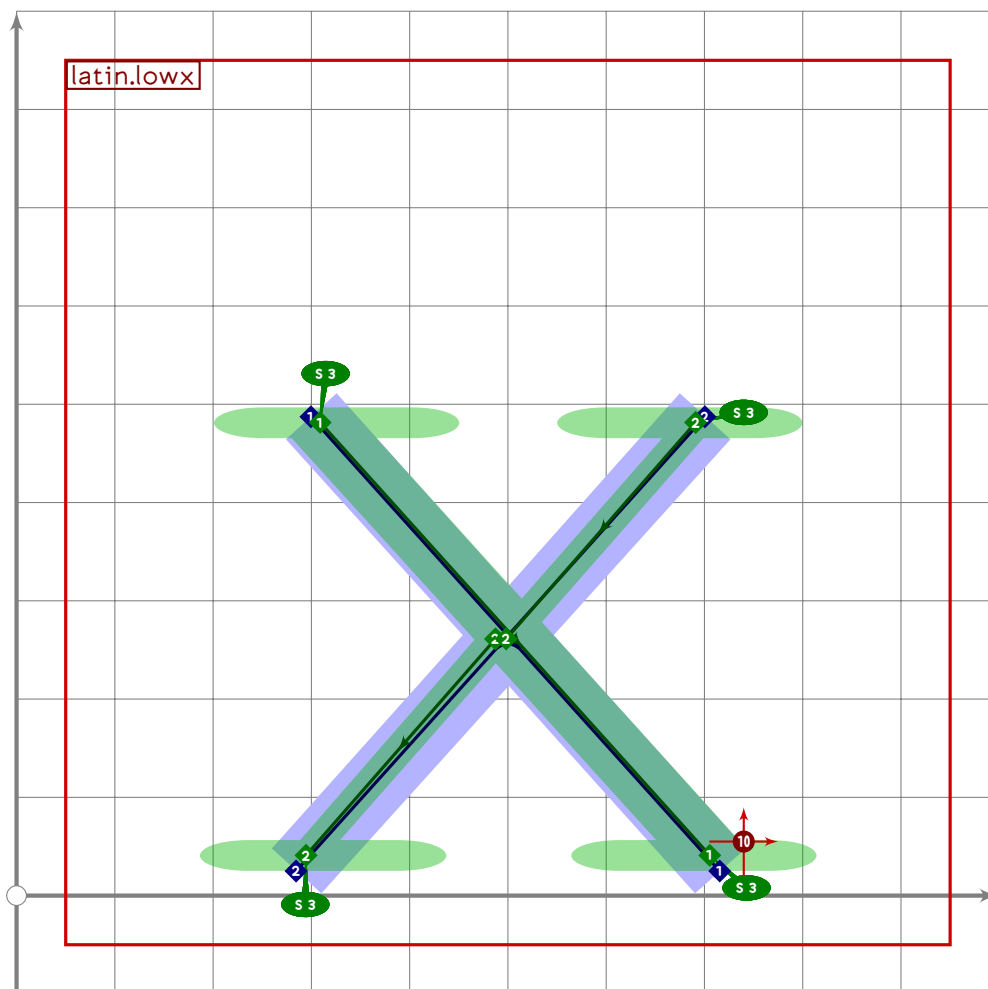
1779
1780 vardef latin.loww =
1781   push_pbox_toexpand("latin.loww");
1782   (x1+x5)/2=(x2+x4)/2=x3=500;
1783   (x3-x2)=(x2-x1);
1784
1785   y1=y3=y5=latin_wide_xheight_v;
1786   y2=y4=latin_wide_low_h;
1787
1788   (x5-x1)=(y1-y2)*1.45;

```

```

1789
1790 if do_alteration:
1791     push_stroke((z1-z2) shifted (alternate_adjust*left),
1792         (1.6,1.6)-(1.6,1.6));
1793     set_boserif(0,0,3);
1794
1795     push_stroke((z2+alternate_adjust*left)-z2-
1796         z3-(z3+alternate_adjust*right),
1797         (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
1798     set_bobrush(0,bralternate);
1799
1800     push_stroke((z3-z4) shifted (alternate_adjust*right),
1801         (1.6,1.6)-(1.6,1.6));
1802
1803     push_stroke((z4+alternate_adjust*right)-
1804         (z4-z5) shifted (alternate_adjust*right*2),
1805         (1.6,1.6)-(1.6,1.6)-(1.6,1.6));
1806     set_boserif(0,2,3);
1807     set_bobrush(0,bralternate);
1808 else:
1809     push_stroke(z1-z2-z3-z4-z5,
1810         (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
1811     set_botip(0,1,0);
1812     set_botip(0,2,0);
1813     set_botip(0,3,0);
1814     set_boserif(0,0,1);
1815 fi;
1816
1817 tsu_accent.shift_anchors(ai=anc_lower_connect)((180,0));
1818 expand_pbox;
1819 endif;

```



```

1820
1821 vardef latin.lowx =
1822   push_pbox_toexpand("latin.lowx");
1823   (x1+x3)/2=500;
1824   (x2+x4)/2=500;
1825   (x2+x3-x1-x4)=((y1-y2)*0.9)*2;
1826   (x3-x1)=(x2-x4)*0.93;
1827
1828   y1=y3=latin_wide_xheight_v;
1829   y2=y4=latin_wide_low_v;
1830
1831   push_stroke(z1-z2,(1.6,1.6)-(1.6,1.6));
1832   set_boserif(0,0,if do_italic_hook: 11 else: 3 fi);
1833   set_boserif(0,1,if do_italic_hook: 11 else: 3 fi);
1834
1835   if do_alteration:
1836     push_stroke(z3-(0.5[z3,z4]+alternate_adjust*right/6)
1837       -(0.5[z3,z4]+alternate_adjust*left/6)-z4,
1838       (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
1839     set_boserif(0,0,if do_italic_hook: 2 else: 3 fi);
1840     set_boserif(0,3,if do_italic_hook: 1 else: 3 fi);

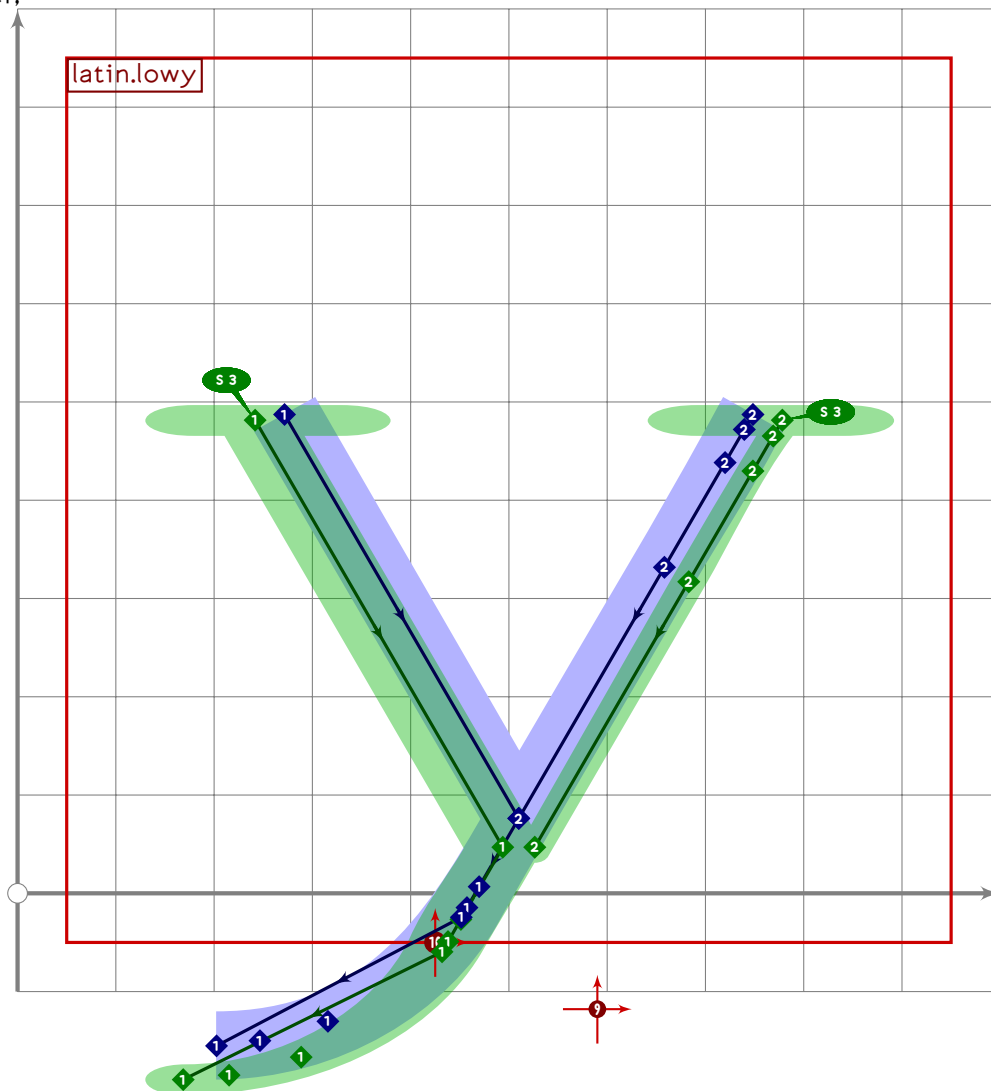
```

U+FF59
tsuku.uniFF59

```

1841 else:
1842     push_stroke(z3-z4,(1.6,1.6)-(1.6,1.6));
1843     set_boserif(0,0,if do_italic_hook: 2 else: 3 fi);
1844     set_boserif(0,1,if do_italic_hook: 1 else: 3 fi);
1845     fi;
1846     set_bobrush(0,bralternate);
1847
1848     tsu_accent.shift_anchors(ai=anc_lower_connect)((240,0));
1849     expand_pbox;
1850 enddef;

```



LATI

```

1851
1852 vardef latin.lowy =
1853     push_pbox_toexpand("latin.lowy");
1854     (x1+x3)/2=(x2+x4)/2=510;
1855     (x2+x3-x1-x4)=((y1-y2)*0.58)*2;
1856     (x3-x1)=(x2-x4)*0.93;
1857     x5=x4-0.1*(x2-x4);
1858

```

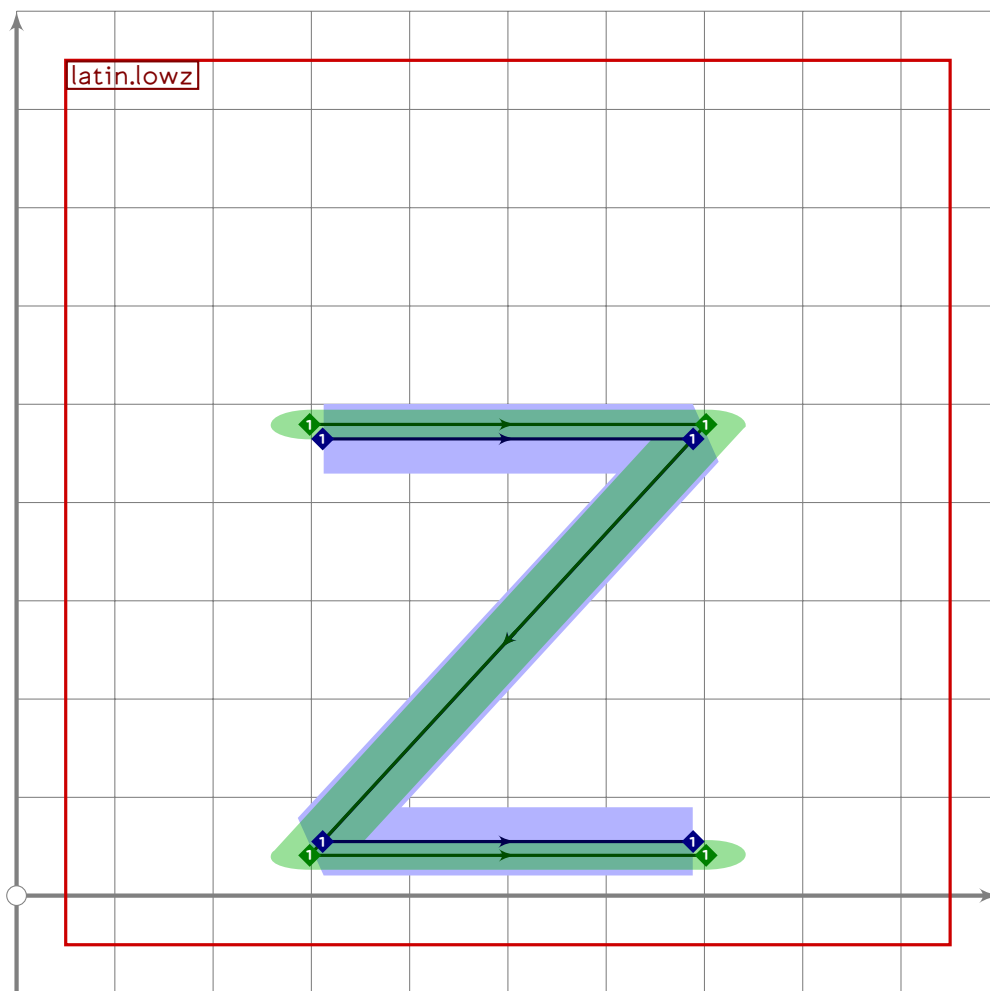


```

1859 y1=y3=latin_wide_xheight_v;
1860 y2=y4;
1861 y5=0.5[y4,latin_wide_low_h]=latin_wide_desc_h;
1862
1863 push_stroke(z1-z2,(1.6,1.6)-(1.6,1.6));
1864
1865 push_stroke(z3..tension 10..(0.6[z3,z4])..tension 0.8 and 3..{left}z5,
1866   (1.6,1.6)-(1.6,1.6)-(1.6,1.6));
1867
1868 numeric xchgtime;
1869 xchgtime:=ypart (get_strokep(-1) intersectiontimes get_strokep(0));
1870
1871 replace_strokep(-1)(z1-subpath (xchgtime,infinity) of get_strokep(0));
1872 replace_strokeq(-1)
1873   ((1.6,1.6)-subpath (xchgtime,infinity) of get_strokeq(0));
1874
1875 replace_strokep(0)(subpath (0,xchgtime) of oldp);
1876 replace_strokeq(0)(subpath (0,xchgtime) of oldq);
1877
1878 set_boserif(-1,0,if do_italic_hook: 11 else: 3 fi);
1879 set_boserif(0,0,if do_italic_hook: 2 else: 3 fi);
1880 set_botip(-1,1,1);
1881 set_bobrush(0,bralternate);
1882
1883 if do_alteration:
1884   replace_strokep(-1)(oldp shifted (alternate_adjust*left/2));
1885   replace_strokep(0)(oldp shifted (alternate_adjust*right/2));
1886 fi;
1887
1888 tsu_accent.shift_anchors(ai=anc_lower)((90,0));
1889 tsu_accent.shift_anchors(ai=anc_lower_connect)((-75,-105));
1890 expand_pbox;
1891 enddef;

```

U+FF5A
tsuku.uniFF5A



```

1892
1893 vardef latin.lowz =
1894   push_pbox__toexpand("latin.lowz");
1895   y1=y2=latin_wide_xheight_h;
1896   y3=y4=latin_wide_low_h;
1897
1898   x1=x3;
1899   x2=x4;
1900   (x1+x2)/2=500;
1901   (x2-x1)=(y1-y3)*0.92;
1902
1903   push_stroke(z1-z2-z3-z4,(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
1904   set_botip(0,1,0);
1905   set_botip(0,2,0);
1906   expand_pbox;
1907 enddef;

```

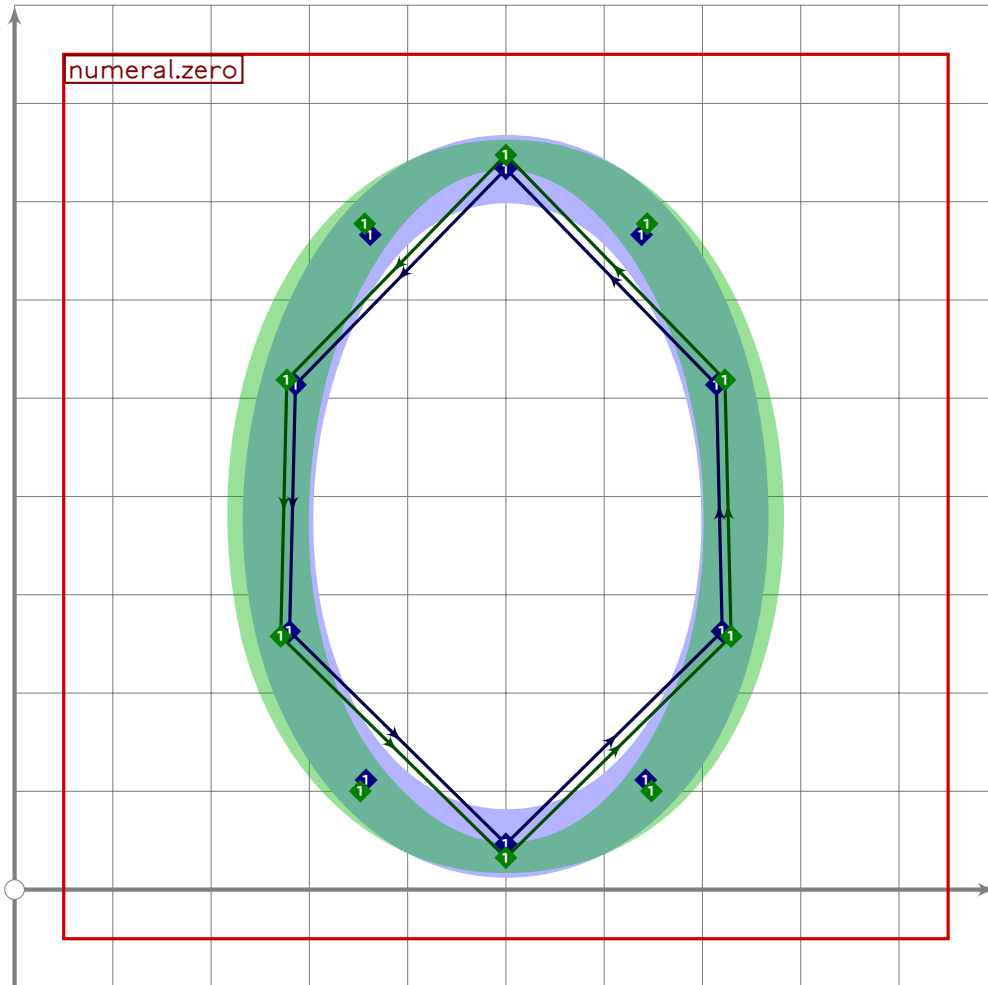
LATI

numerals.mp

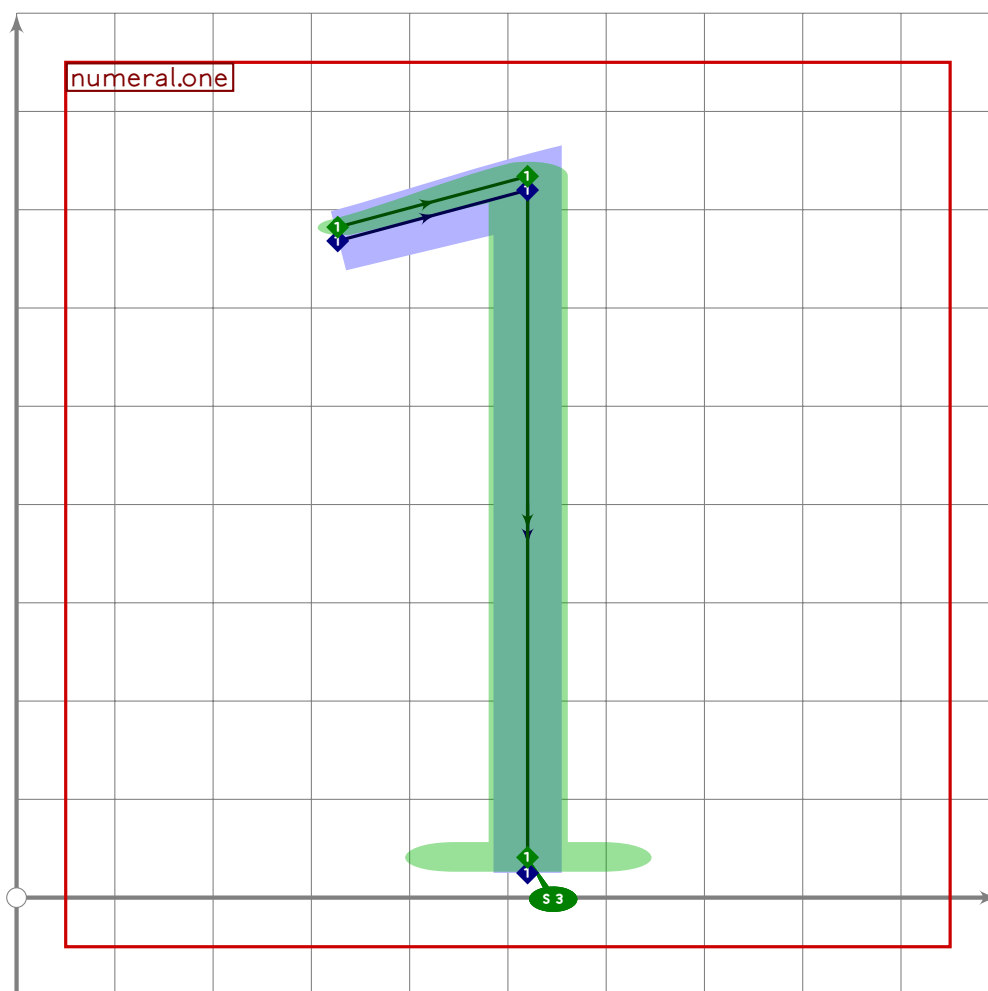
```

1 %
2 % Hindu/Arabic numerals for Tsukurimashou
3 % Copyright (C) 2011, 2013 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;

```



U+FF11
tsuku.uniFF11

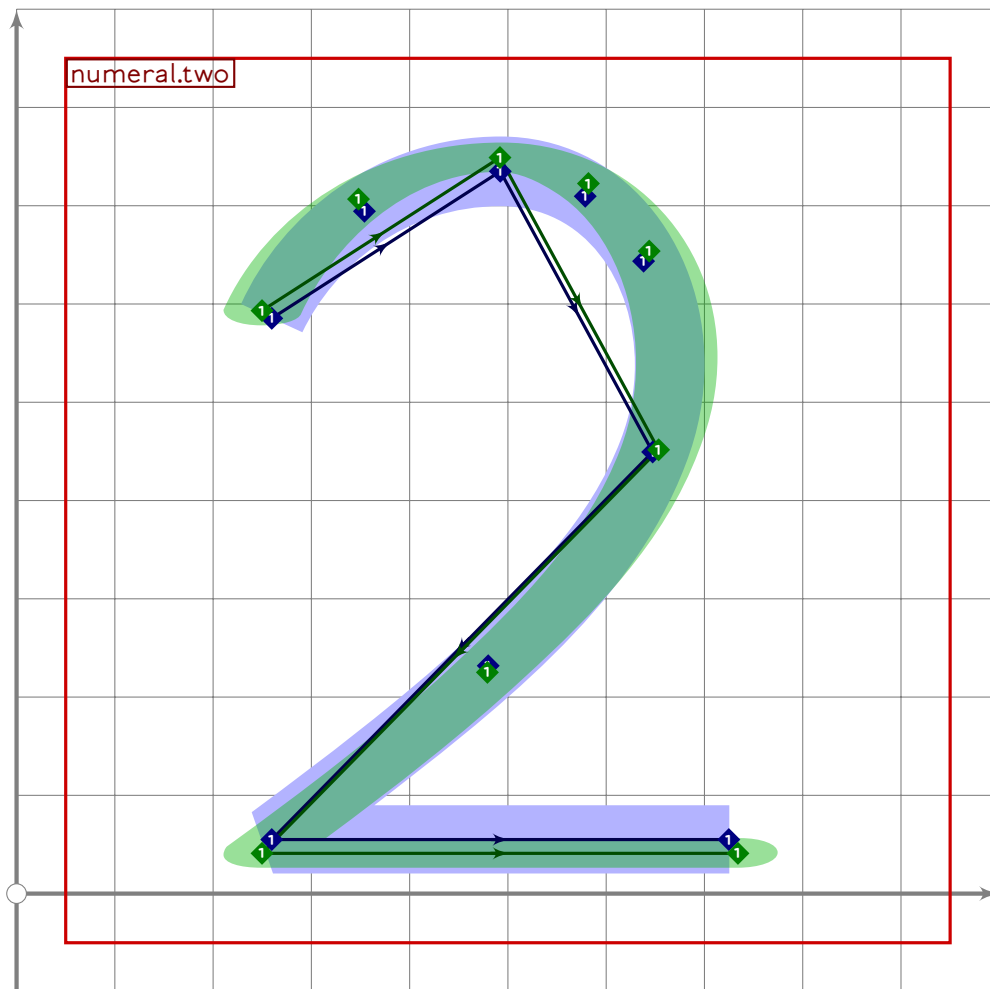


```

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;

```

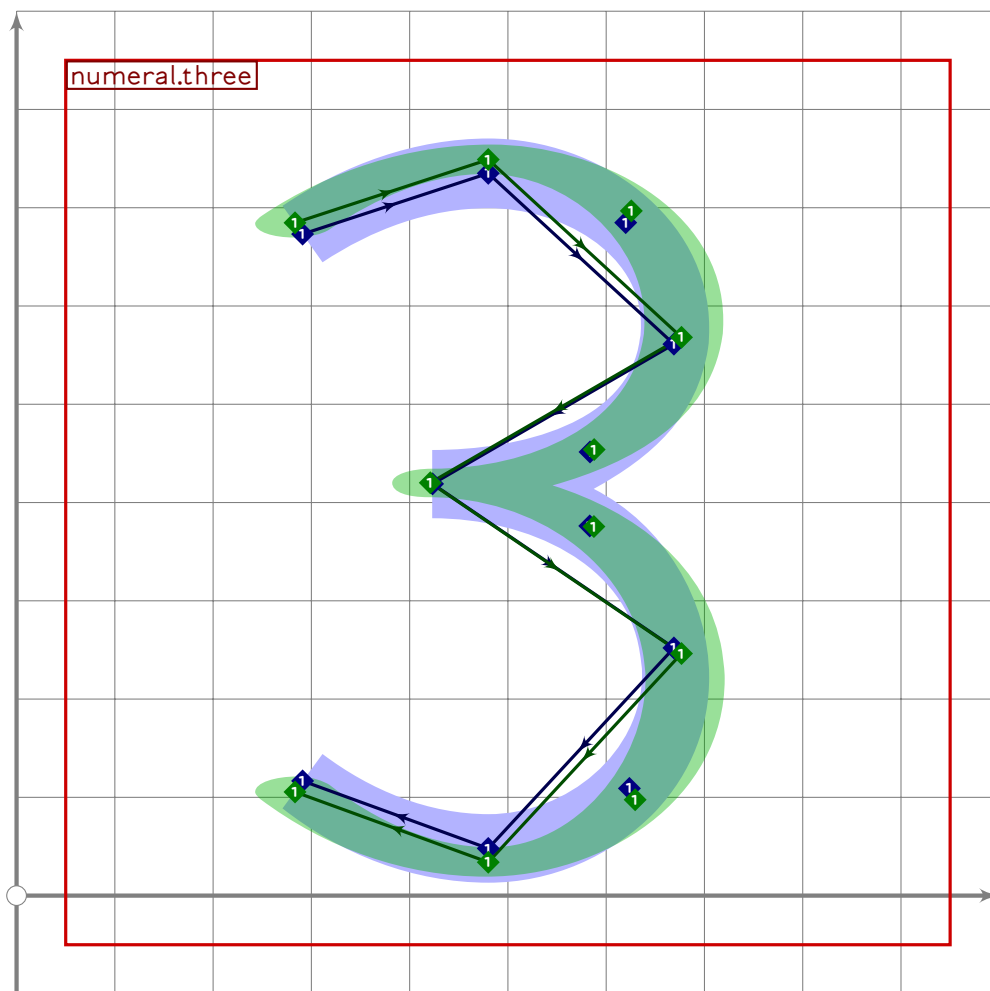
NUME



```

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;

```



```

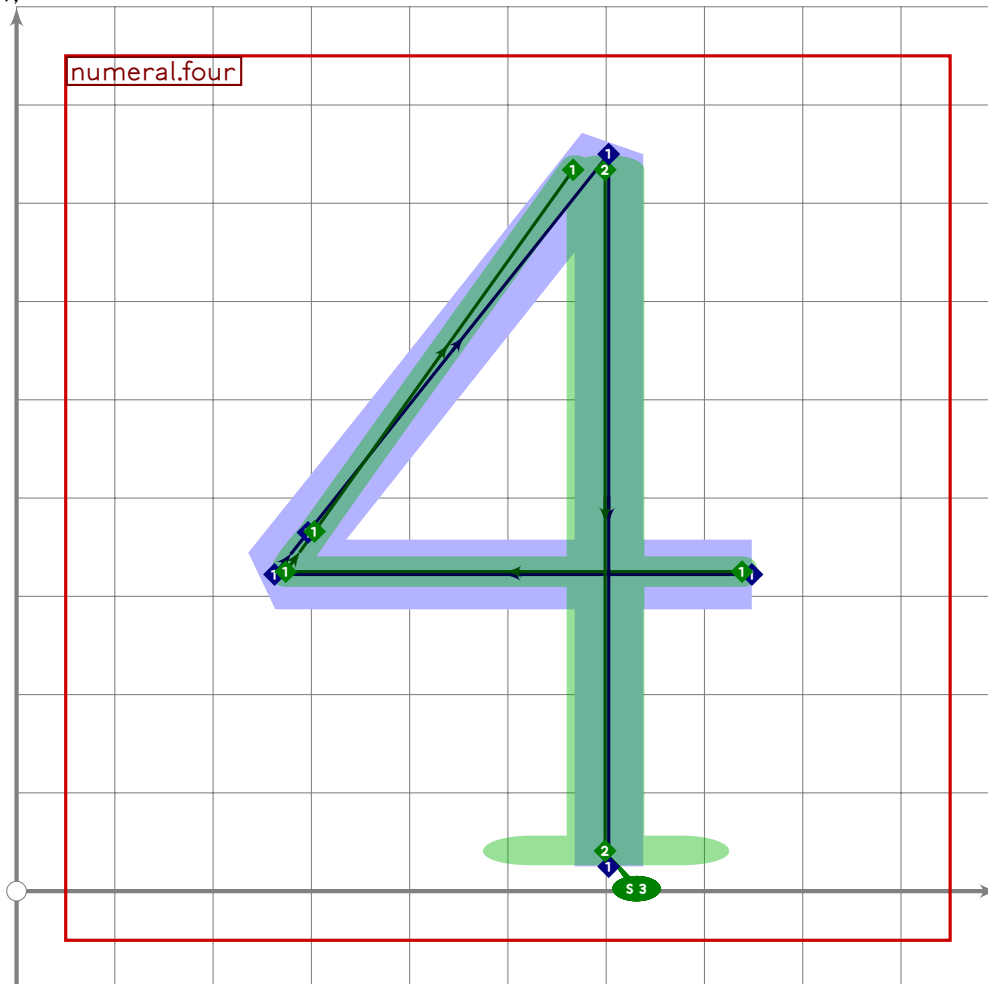
78
79 vardef numeral.three =
80   push_pbox__toexpand("numeral.three");
81   x1=x7;
82   x2=x6=0.5[x1,x3];
83   x3=x5;
84   x4=0.35[x1,x3];
85   (x1+x3)/2=480;
86   (x3-x1)=0.55*(y2-y6);
87
88   y1=0.91[y6,y2];
89   y2=latin_wide_high_r;
90   y3=0.45[y4,y2];
91   y4=0.54[y6,y2];
92   y5=0.45[y4,y6];
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)-

```

```

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);

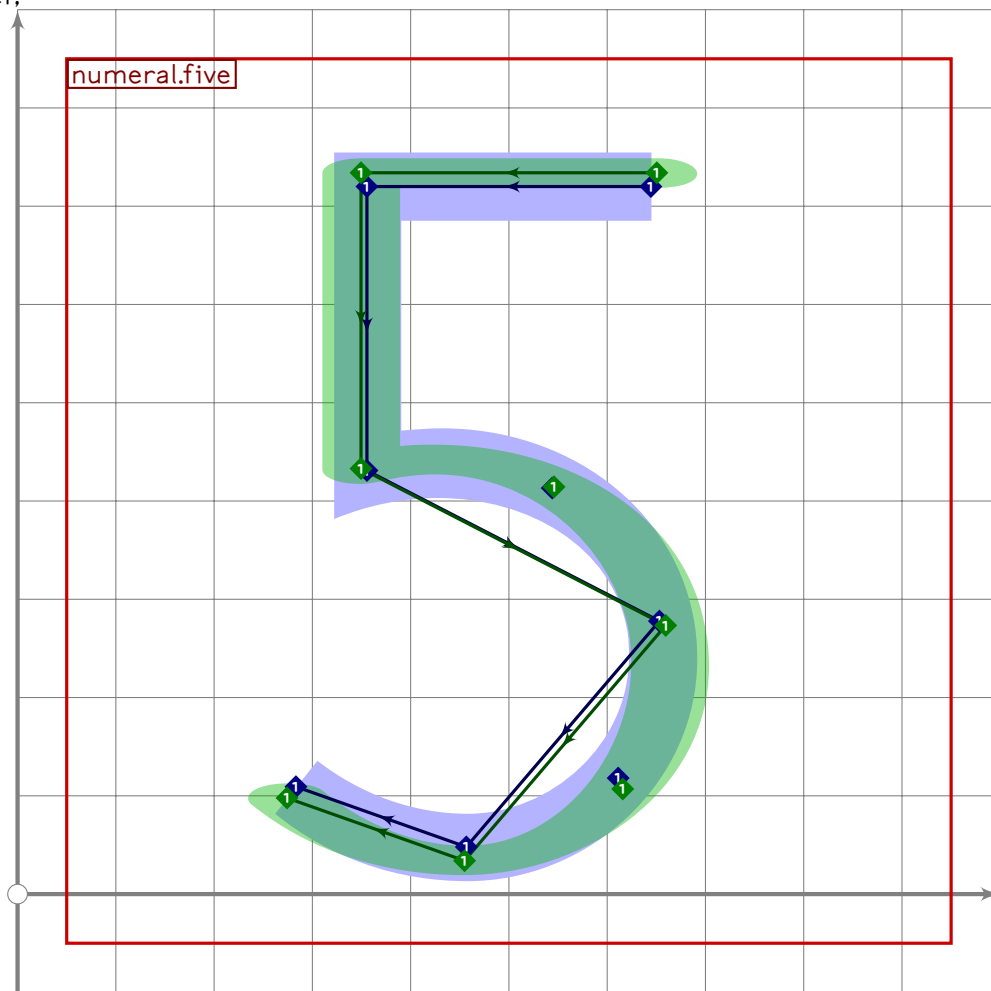
```

U+FF15
tsuku.uniFF15

```

120   set_bobrush(0,bralternate);
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;

```



NUME

```

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];

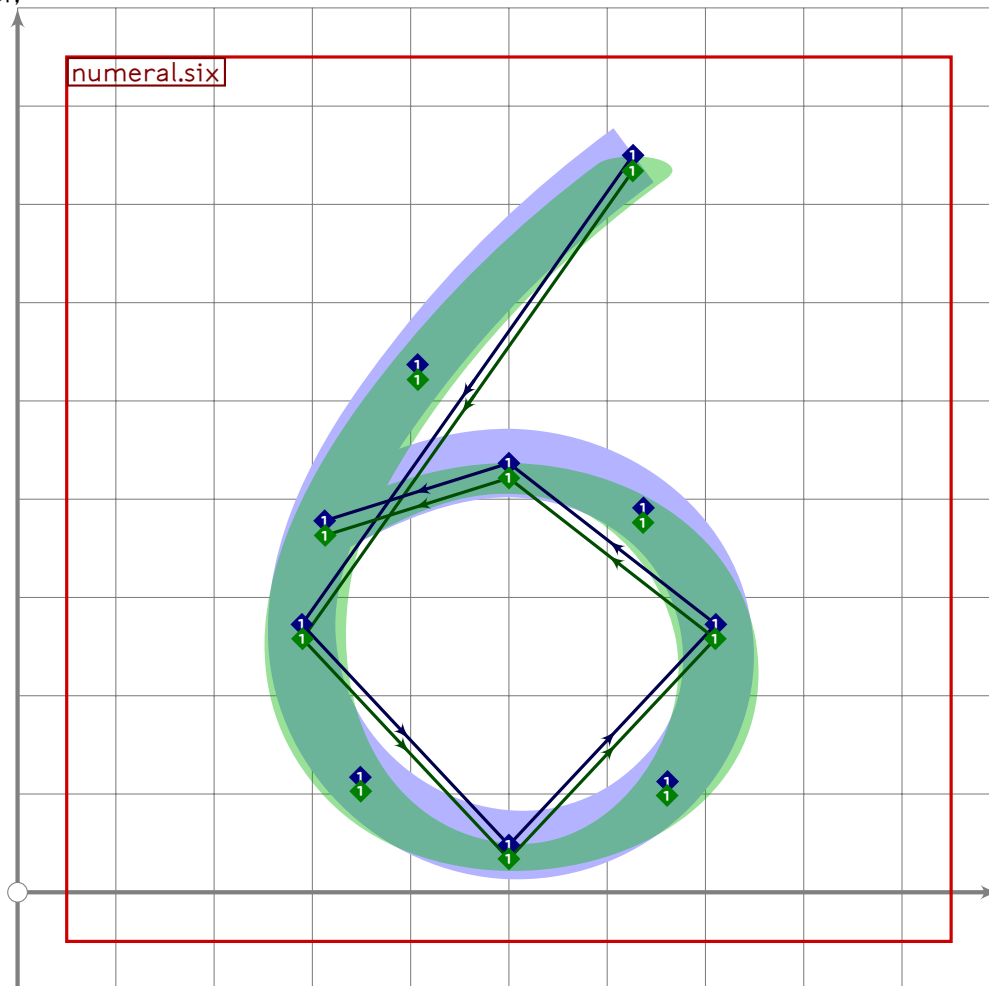
```



```

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);
153 set_botip(0,2,1);
154 expand_pbox;
155 enddef;

```



```

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);

```

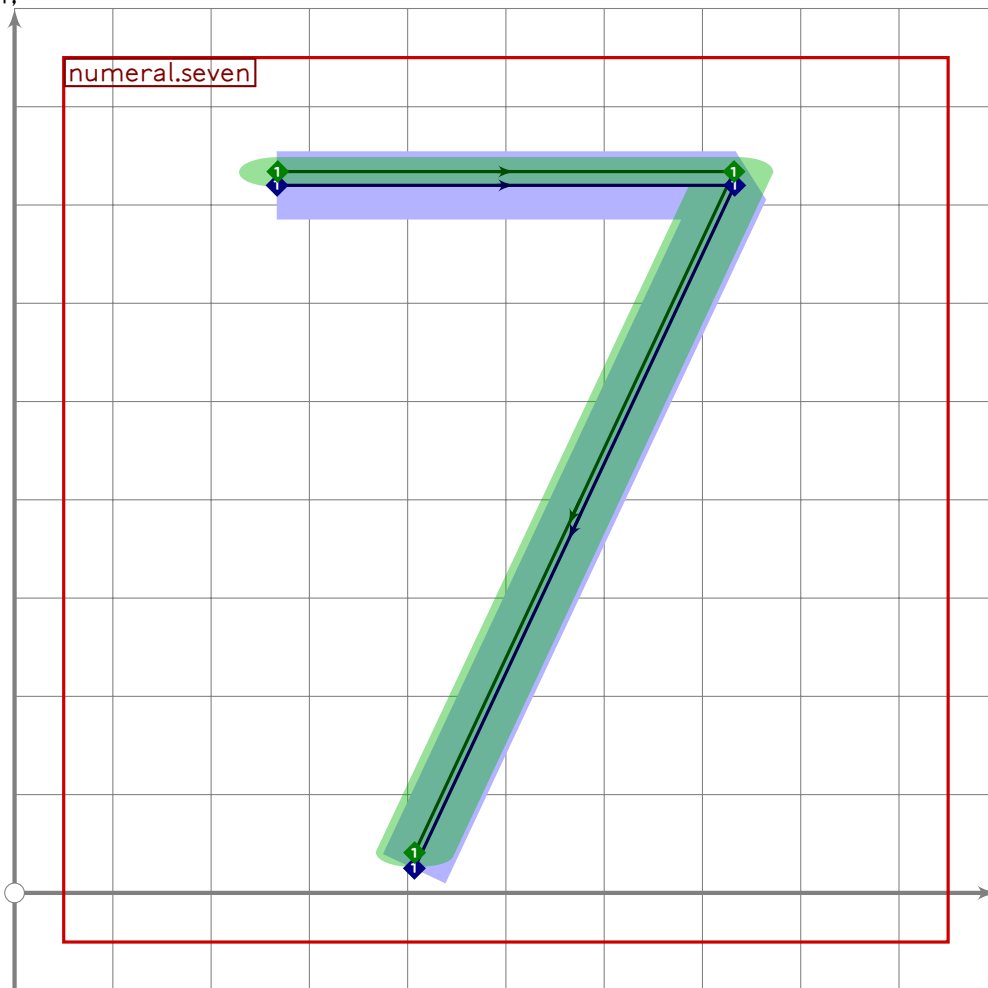
NUME

U+FF17
tsuku.uniFF17

```

162 x5=x3;
163
164 y1=latin_wide_high_v;
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 enddef;

```



```

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;

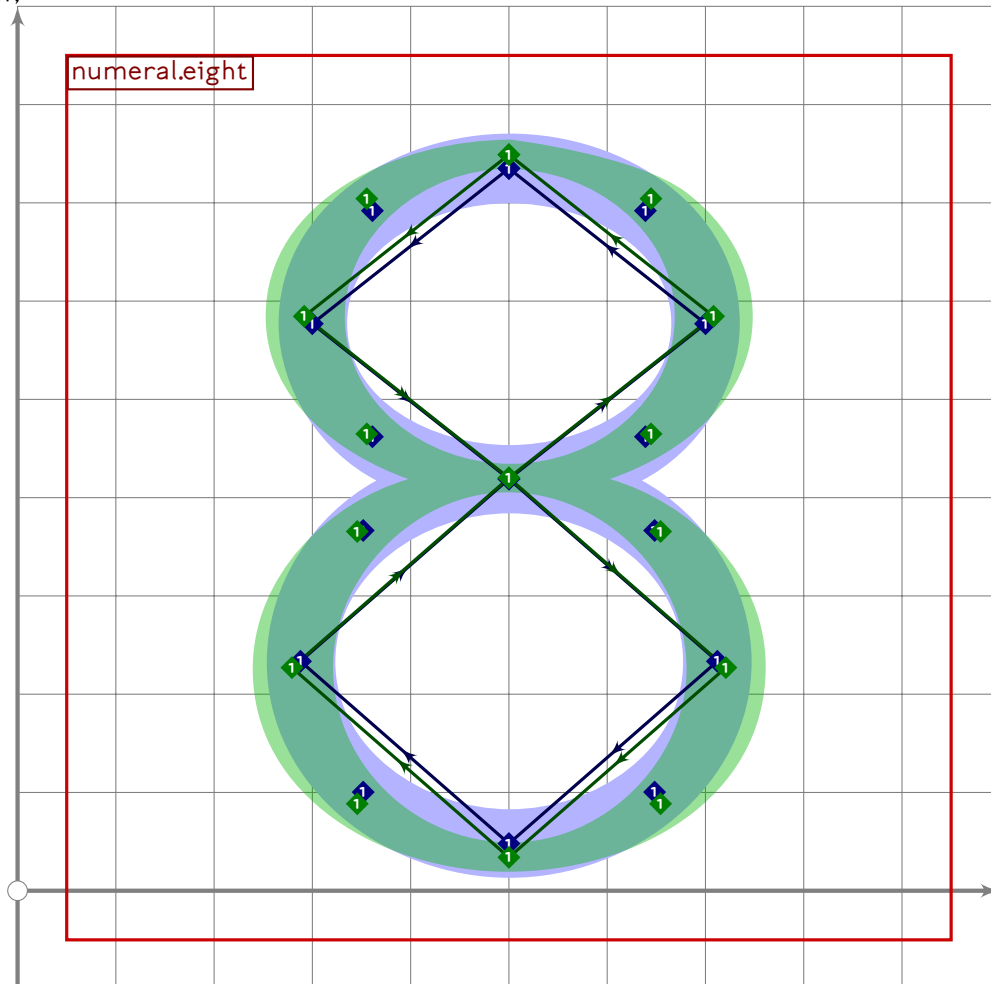
```

NUME

```

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,

```

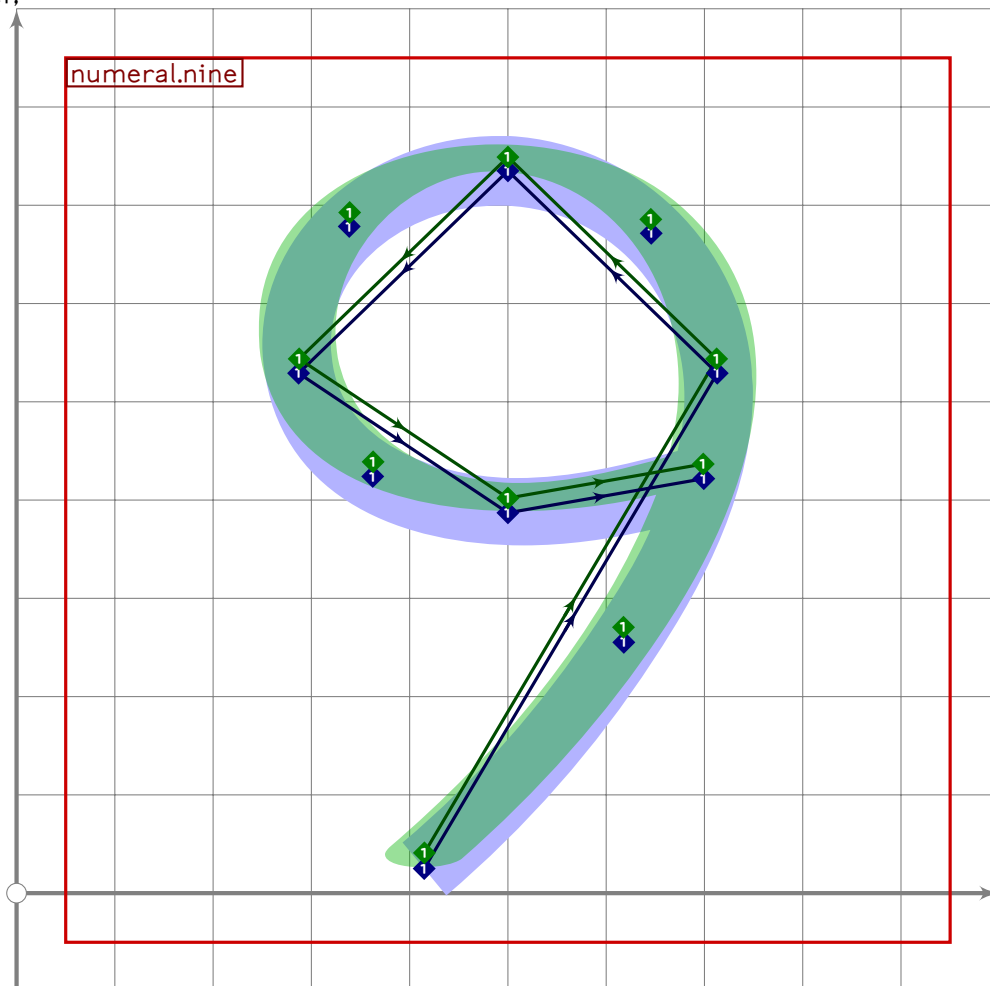
NUME

U+FF19
tsuku.uniFF19

```

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));

```

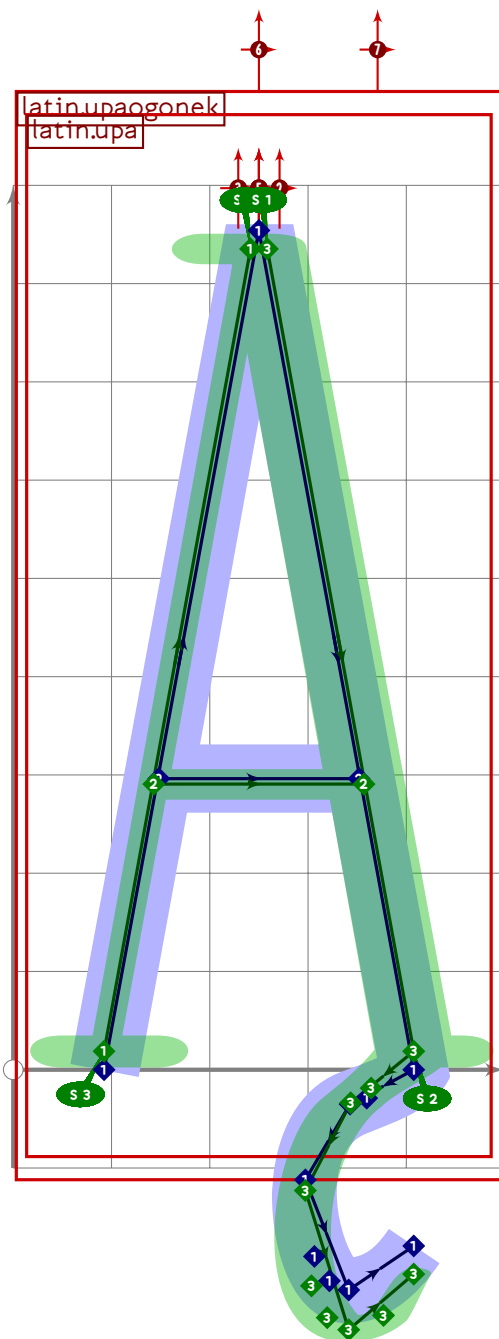
NUME

```
225   expand__pbox;  
226 endif;
```

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
```

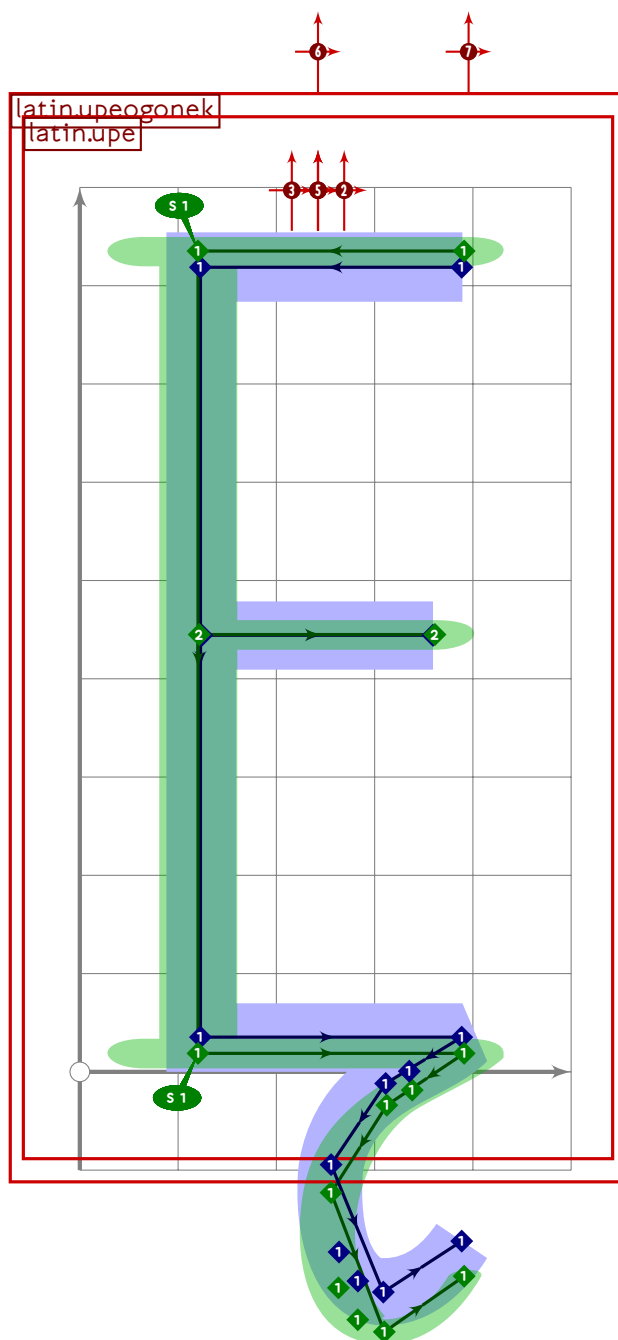


OGON

```

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-(14,14)-(1.3,1.3)-(14,14)-(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-(14,14)-(1.3,1.3)-(14,14)-(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];

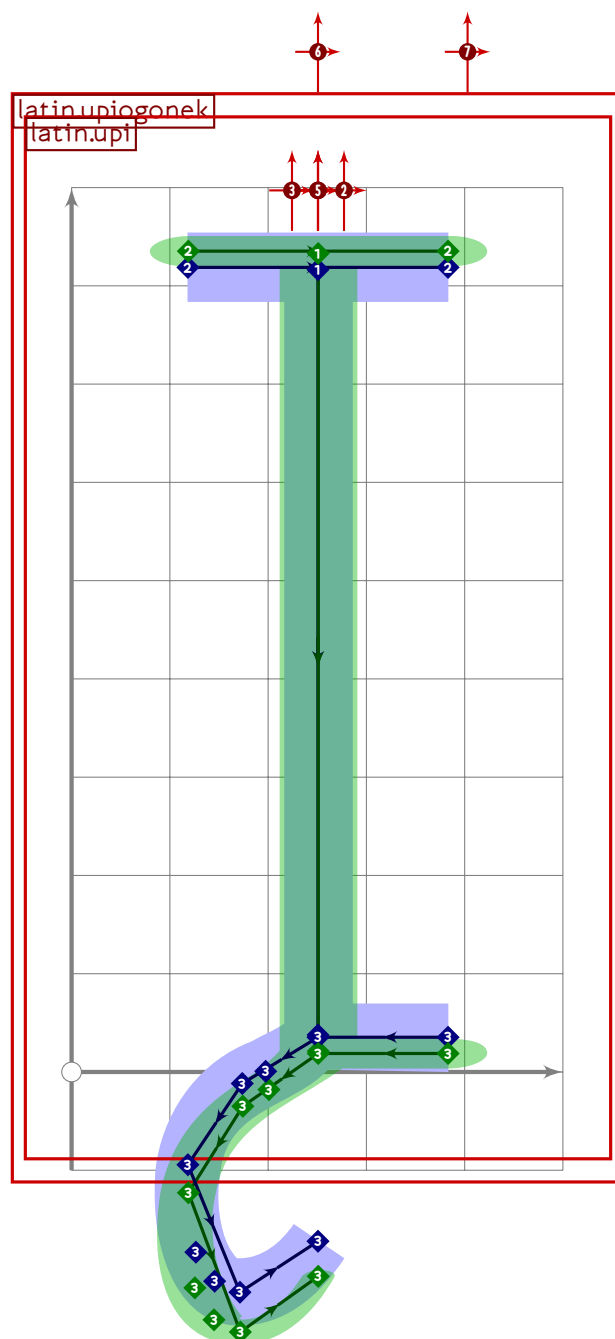
```



```

74
75 replace_strokep(-1)(oldp{dir 210}.z7..z8{right}.z9);
76 replace_strokep(-1)(insert_nodes(oldp)(length(oldp)-2.5));
77 replace_strokeq(-1)(oldq-(14,14)-(1.3,1.3)-(14,14)-(1,1));
78 set_botip(-1,length(get_strokep(-1))-4,0);
79 expand_pbox;
80 enddef;

```



```

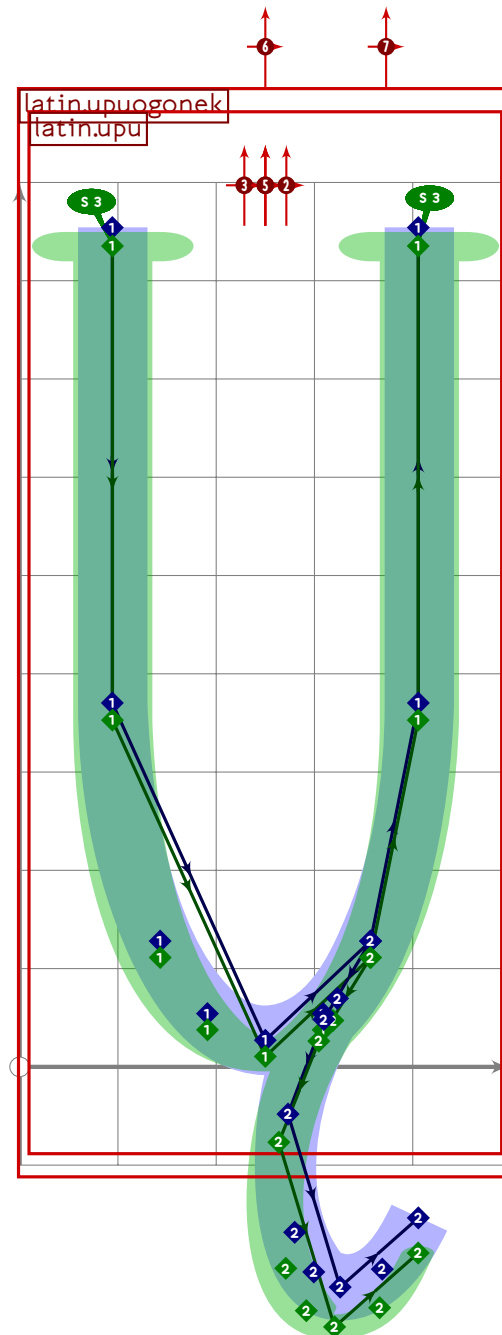
81
82 vardef latin.upiogonek =
83   push_pbox_toexpand("latin.upiogonek");
84   latin.upi;
85

```

```

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)-(14,1.4)-
98   (1.3,1.3)-(14,1.4)-(1,1));
99 expand_pbox;
100 enddef;

```



```

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

```

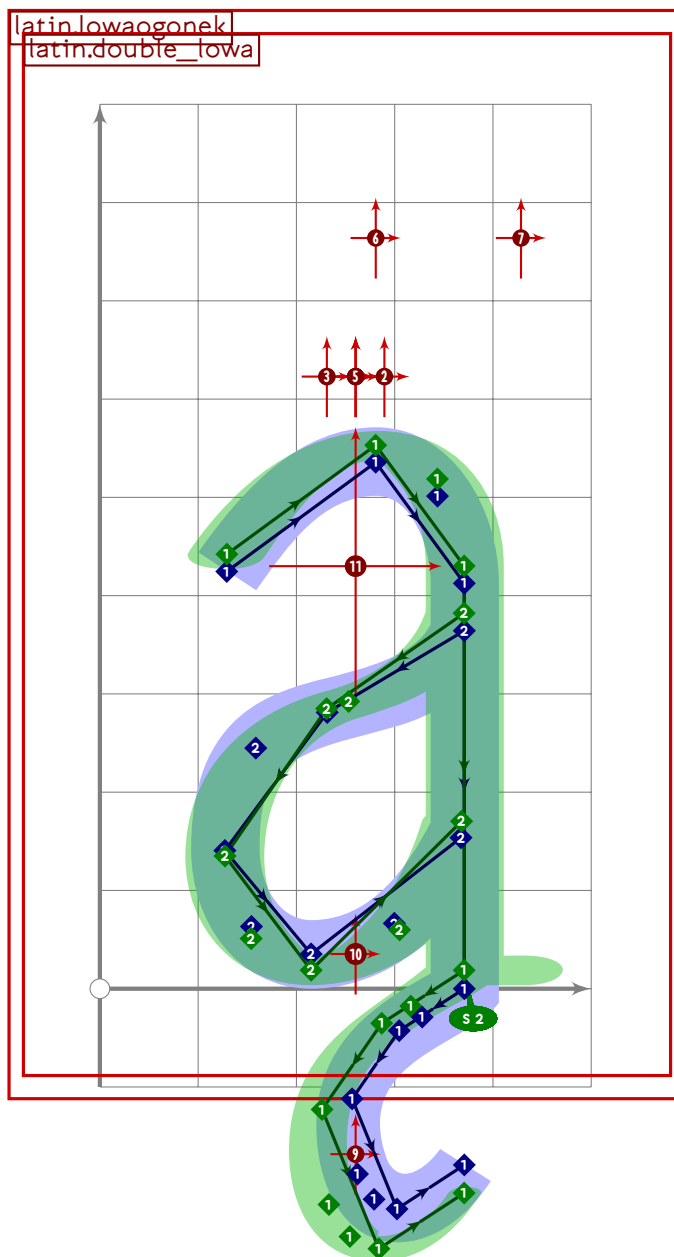
U+0105

tsuku.aogonek

```

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)-(1,4,1)-(1,3,1)-(1,4,1)-(1,1));
123 replace_stroke(0,insert_nodes(oldp)(length(oldp)-2.5));
124 expand_pbox;
125 endif;

```



OGON

126

```

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-(14,14)-(1.3,1.3)-(14,14)-(1,1));
142   set_botip(-1,length(get_strokep(-1))-4,1);
143   expand_pbox;
144 enddef;

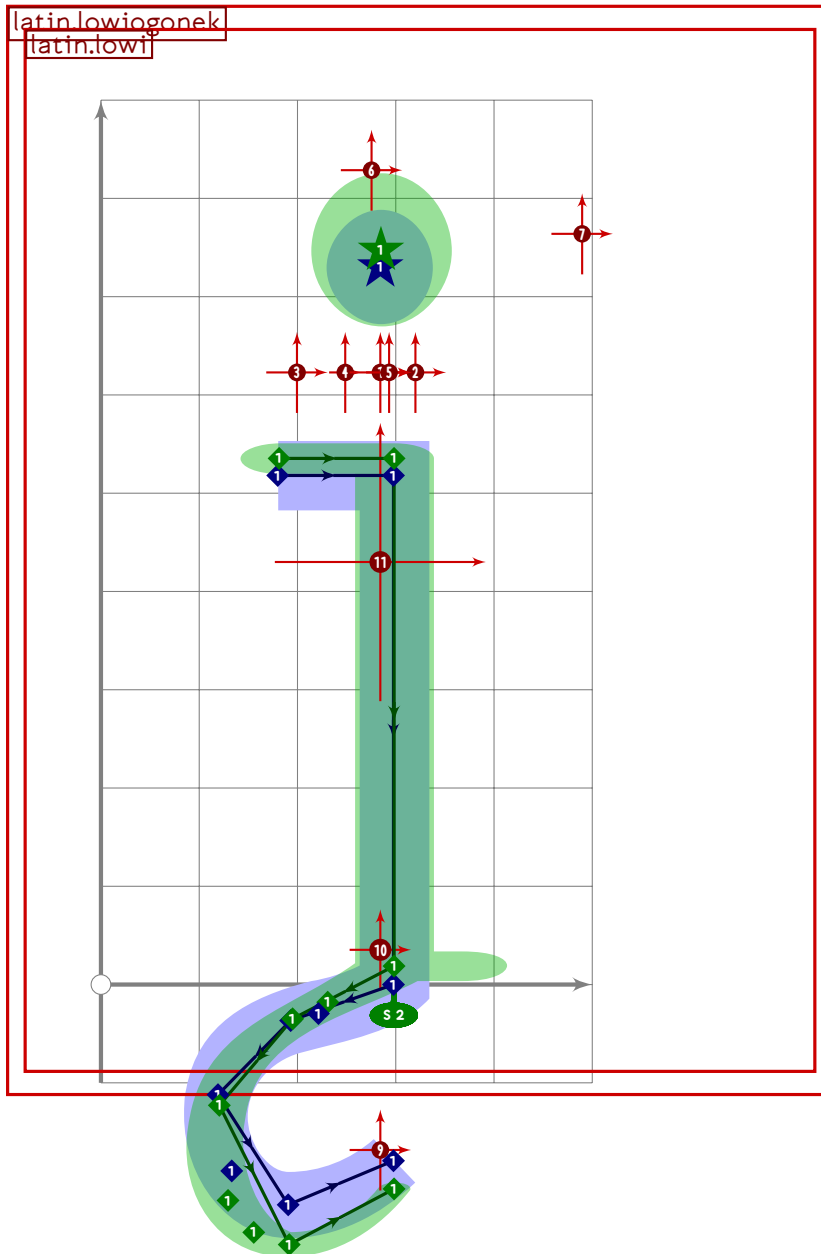
```

OGON

```

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)-(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;

```



```

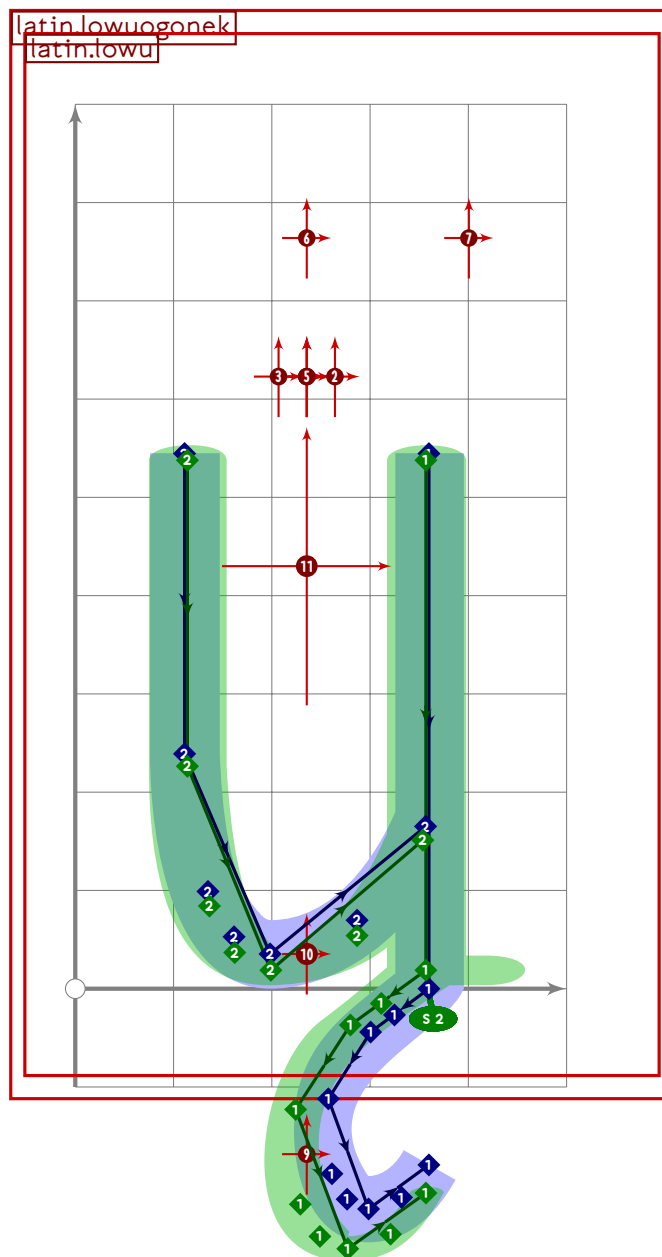
166
167 vardef latin.lowiogonek =
168 push_pbox_toexpand("latin.lowiogonek");
169 latin.lowi;
170
171 x5=x7-200;
172 x6=0.4[x5,x7];

```

U+0173

tsuku.uogonek

```
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-(14,14)-(1.3,1.3)-(14,14)-(1,1));
182 set_boserif(0,2,2);
183 set_botip(0,2,1);
184 expand_pbox;
185 endif;
```



OGON

186


```

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;

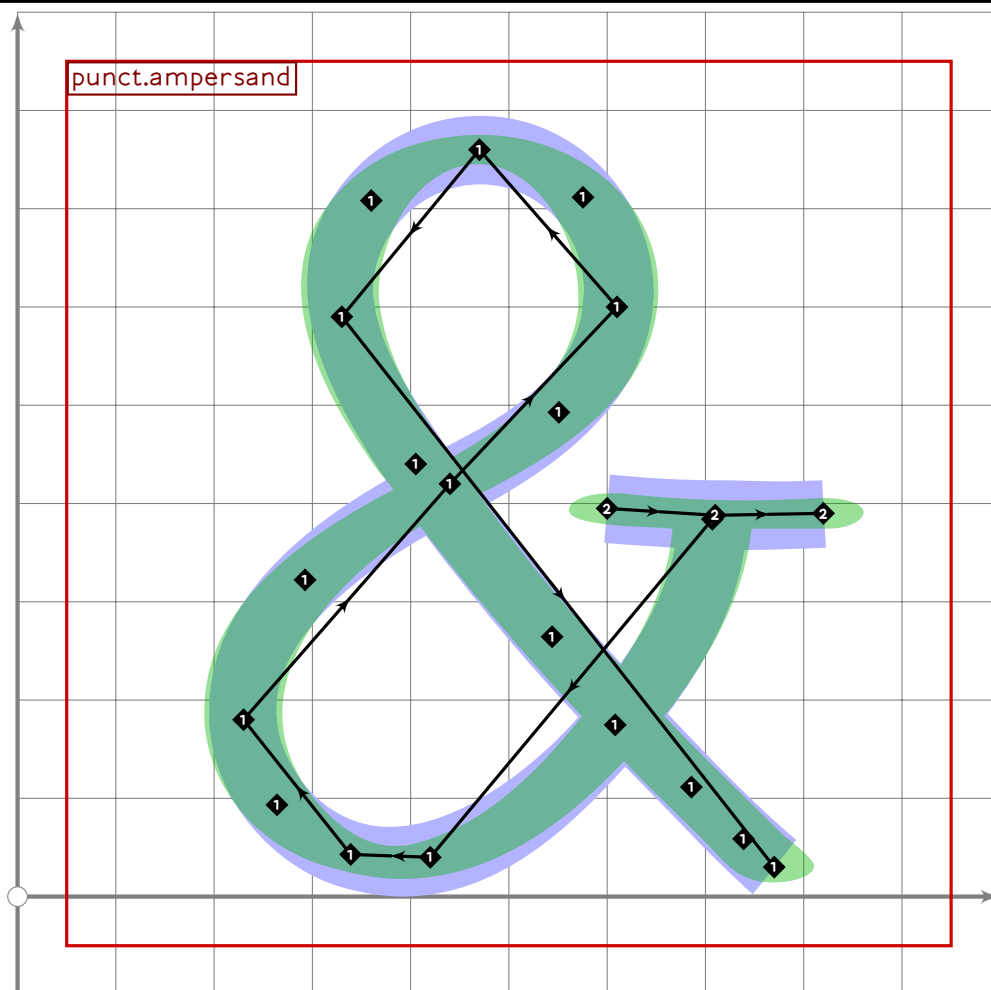
```

punct.mp

```

1 %
2 % Punctuation for Tsukurimashou
3 % Copyright (C) 2011, 2013, 2015 Matthew Skala
4 %
5-29 [Standard copyright notice]
30
31 inclusion_lock(punct);
32
33

```



```

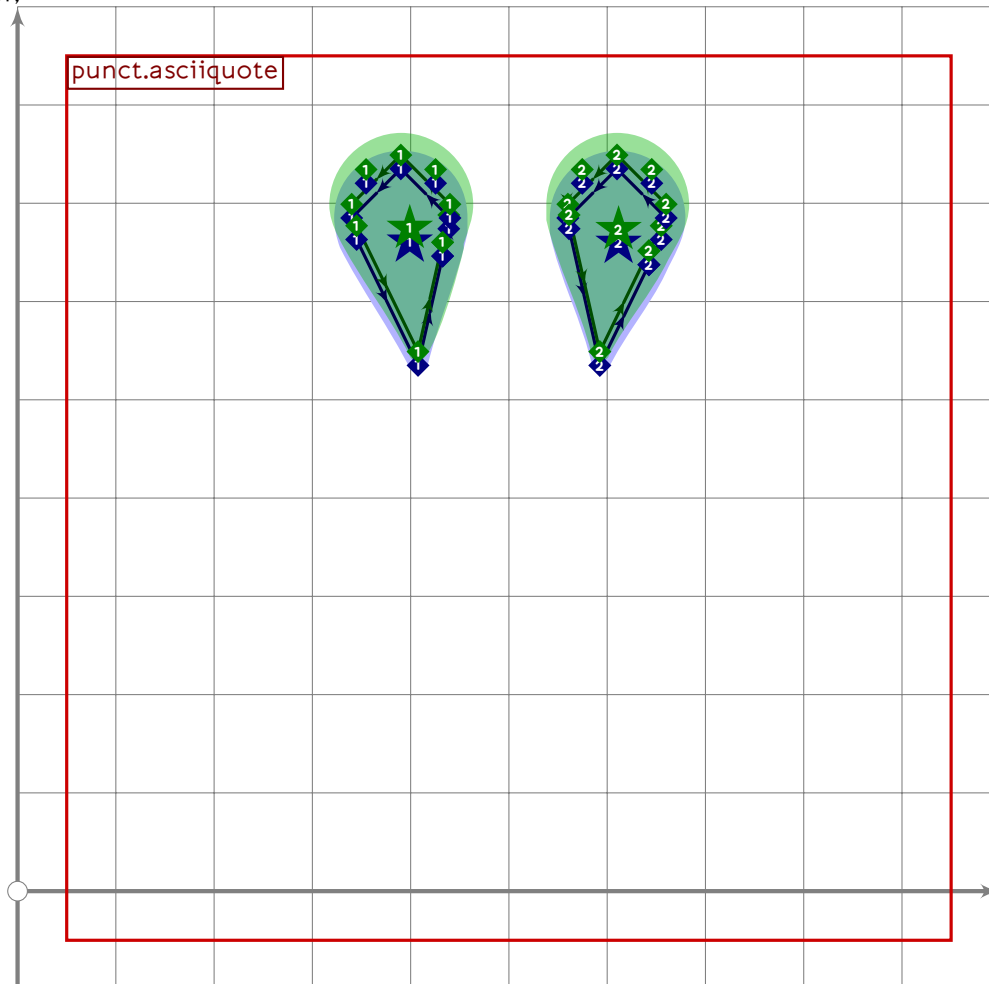
34
35 vardef punct.ampersand =
36   push_pbox_toexpand("punct.ampersand");
37
38   push_stroke((707,384)..tension 1.3..(420,40)..(230,180)..(440,420)..
39     (610,600)..(470,760)..(330,590)..tension 1.5 and 4..(770,30),
40     (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-
41     (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6));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

```

```

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;

```



```

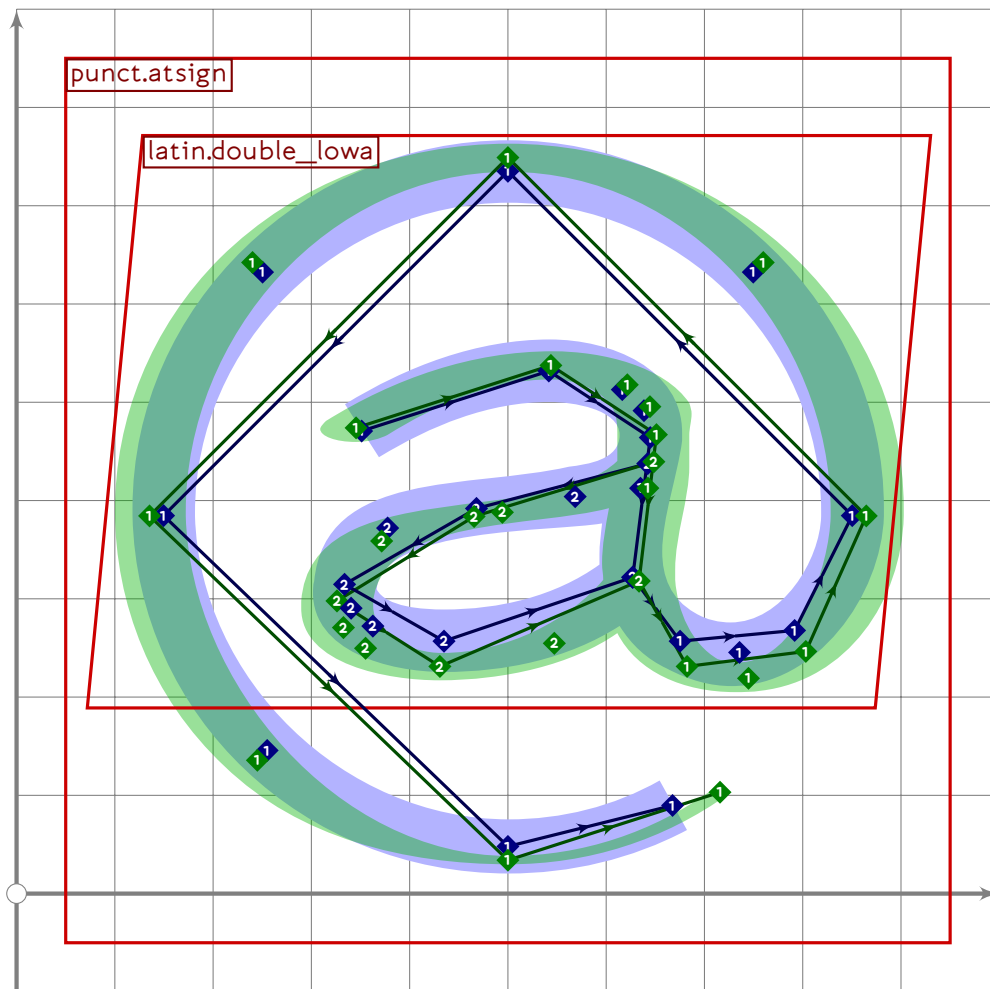
48
49  vardef punct.asciquote =
50    push_pbox_toexpand("punct.asciquote");
51
52    numeric dx;
53    dx=tsu_punct_size;
54
55    (x1+x2)/2=(x3+x4)/2=500;
56    x2-x1=1.2*dx+tsu_punct_size;
57    x4-x3=tsu_punct_size*1.85;
58
59    y1=y2=latin_wide_high_r-dx/2;
60    y3=y4=y1-1.5*dx;
61
62    path 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_bobrush(0,brpunct);
73  set_bosize(0,75);
74  set_botip(0,6,0);
75
76  push_stroke((down..right..up..left..cycle) scaled (dx/2) shifted z2,
77    (2,2)-(2,2)-(2,2)-(2,2)-(2,2)-(2,2)-(1.3,1.3)-cycle);
78  replace_strokep(0)(z4-(subpath ((xpart (oldp intersectiontimes
79    (ptmp shifted z4))),4-xpart ((reverse oldp) intersectiontimes
80    (ptmp shifted z4)))) of oldp)-cycle);
81  replace_strokep(0)((subpath (0,8,6) of oldp)-cycle);
82  set_bobrush(0,brpunct);
83  set_bosize(0,75);
84  set_botip(0,6,0);;
85
86  if tsu_brush_max.brpunct>=0.3:
87    push_lcblob(get_strokep(-1));
88    push_lcblob(get_strokep(0));
89  fi;
90  expand_pbox;
91 enddef;

```



```

92
93 vardef punct.atsign =
94   push_pbox__toexpand("punct.atsign");
95   begingroup
96     save xsp,ysp;
97     xsp:=sp;
98     latin.low_a;
99     set_boserif(-1,3,whatever);
100    ysp:=sp;
101
102    numeric x[],y[];
103    x1-x2=x2-x3=y2-y1;
104    x2=x4=500;
105    y1=y3=0.49[y4,y2];
106    y2=latin_wide_high_r;
107    y4=latin_wide_low_r;
108
109    transform shrinka;
110    (0.5[lcorner get_stroke(-1),urcorner get_stroke(-1)])
111      transformed shrinka=0.5[z3,z1];
112    (0.5[lcorner get_stroke(-1),urcorner get_stroke(-1)])

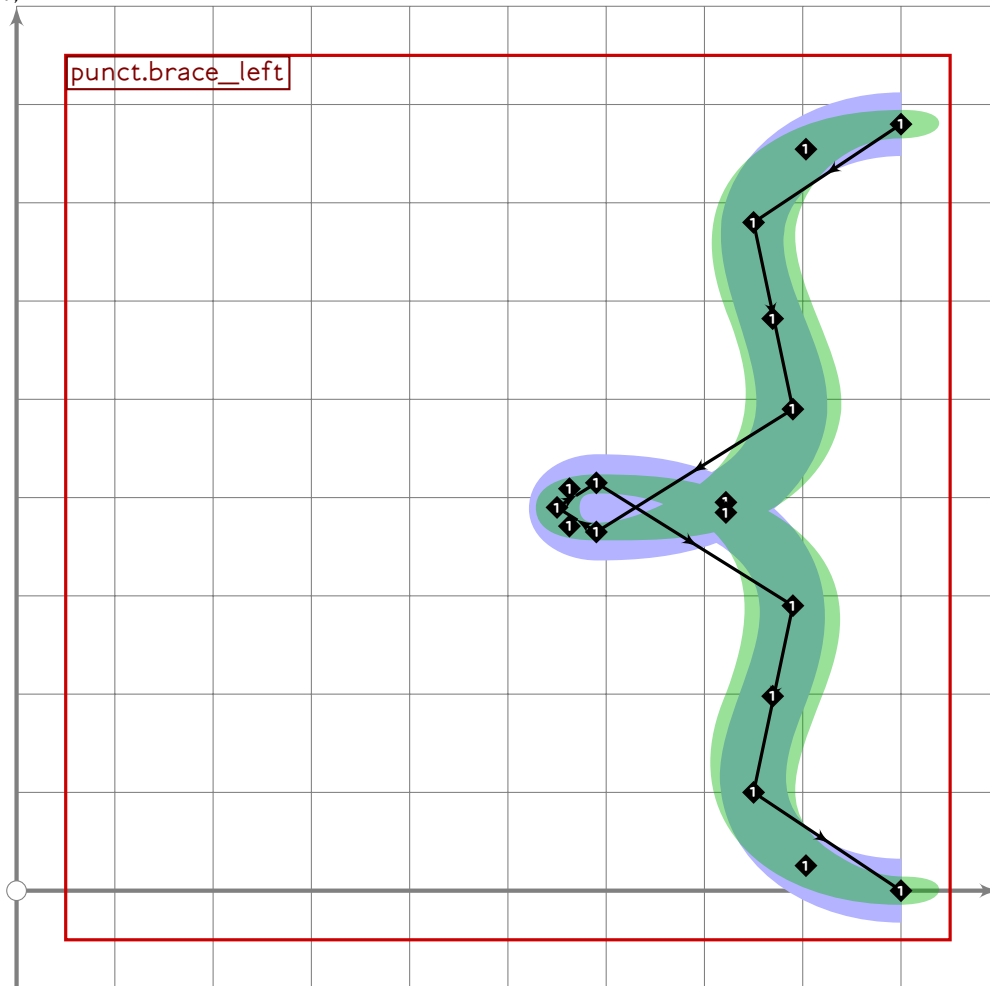
```

U+FF5B
tsuku.uniFF5B

```

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

```



PUNC

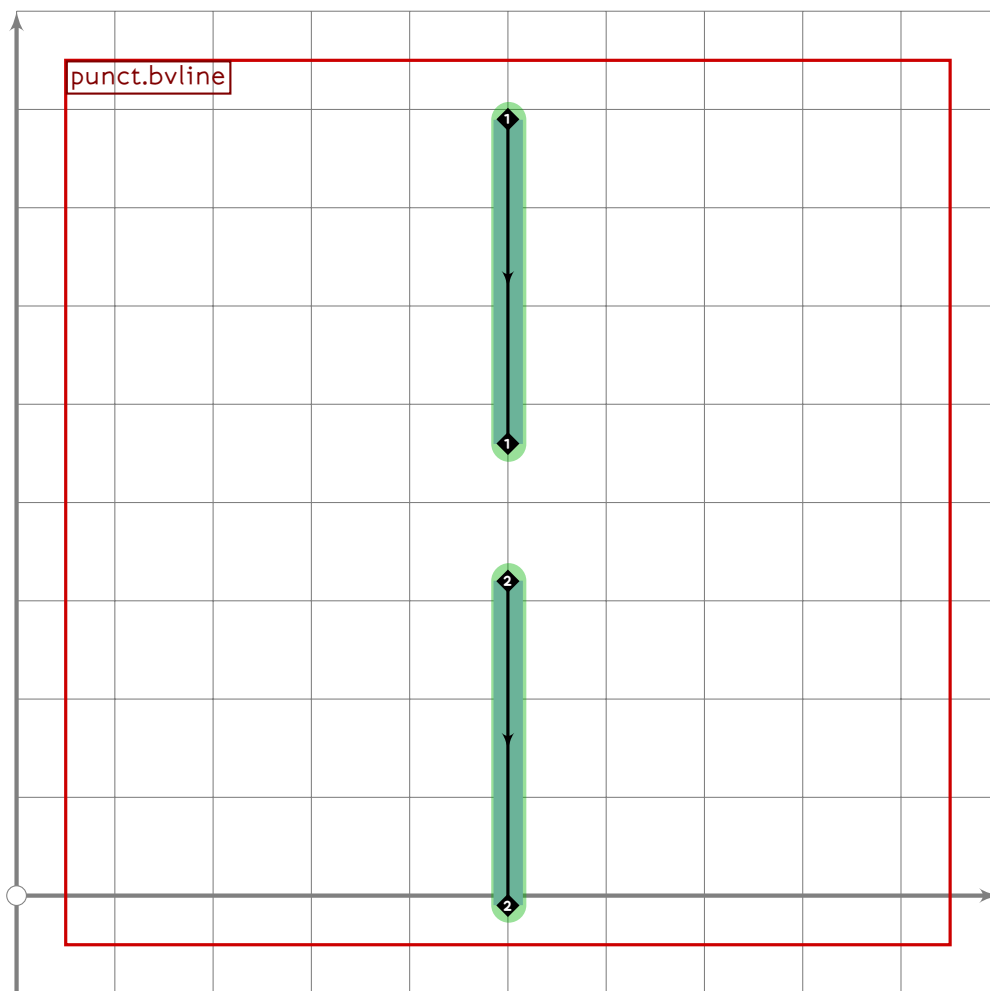
```

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

```

A 2D plot on a grid showing a green region with a black path. The path starts at the bottom left, goes up, then right, then up again, and finally right to the top right. The path is marked with black diamonds and arrows. The green region is surrounded by a red bounding box. The plot includes axes and a grid.

PUNC



```

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

```



```

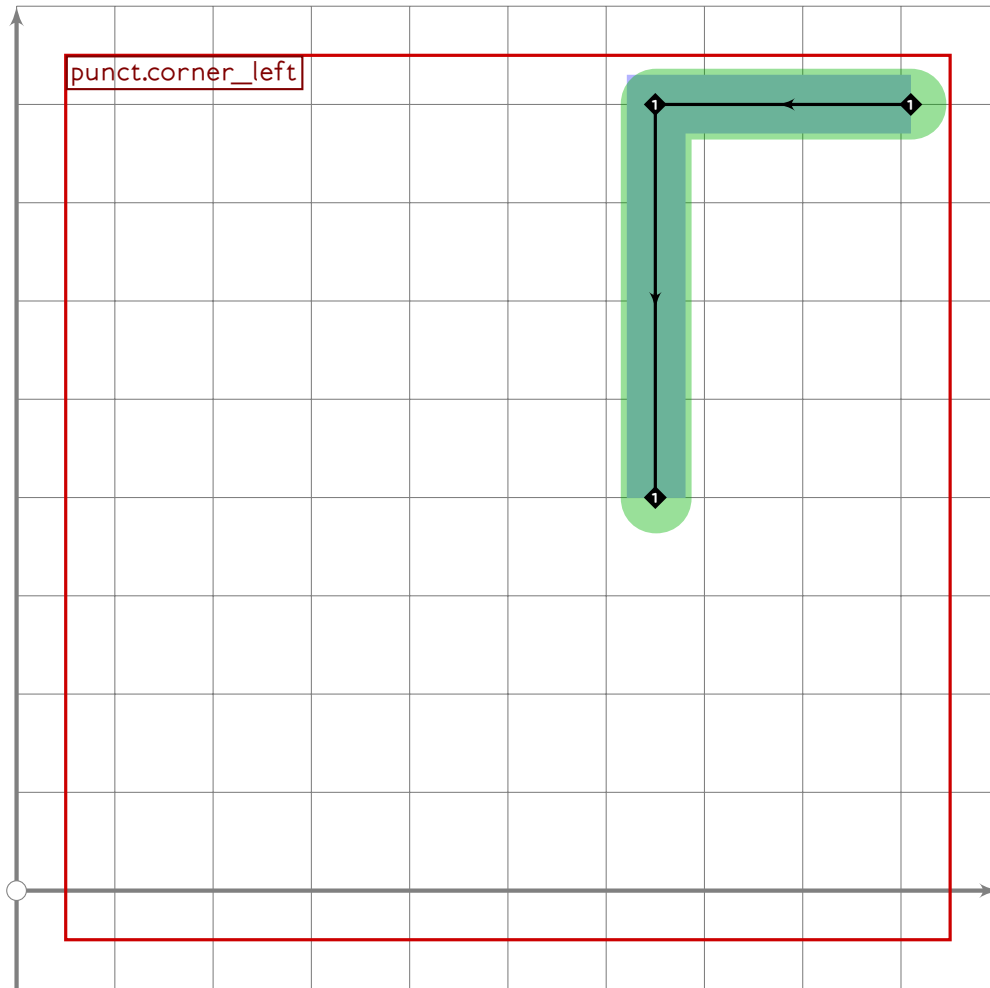
175 xsp:=sp;
176 sp:=1;
177 t:=tsu_rescale_xform;
178 sp:=xsp;
179
180 x1=0.8[x2,x4];
181 (x2+x4)/2=x3=0;
182 (x2-x4)=0.45*(y3-y1)=tsu_punct_size;
183 x5=x3;
184
185 y2=y4=0.32[y3,y1]=0;
186 y5-y4=0.73*(y4-y3);
187
188 push_stroke(z1{curl 0.2}..tension 1.2..z2..z3..z4{dir 280}..
189   z5..{curl 0.2}(point 0.8 of (z1{curl 0.2}..tension 1.2..
190   z2..z3..z4{dir 280})),
191   (2,2)-(2,2)-(2,2)-(2,2)-(2,2)-(2,2)-(2,2));
192 replace_strokep(0)((point 4.2 of oldp)-oldp);
193 (0,0) transformed u=llcorner get_strokep(0);
194 (1,0) transformed u=lrcorner get_strokep(0);
195 (0,1) transformed u=ulcorner get_strokep(0);
196 replace_strokep(0)(oldp rotated (cang-6) shifted (cpos transformed t)
197   transformed inverse t);
198 u:=u scaled 1.3 rotated (cang-6) shifted (cpo transformed t)
199   transformed inverse t;
200 set_botip(0,1,0);
201 set_bobrush(0,brpunct);
202
203 if tsu_brush_max.brpunct>=0.3:
204   set_bosize(0,40);
205   push_lcblob(get_strokep(0)-cycle);
206 else:
207   replace_strokep(0)(subpath (0,5.2) of oldp);
208   set_bosize(0,80);
209 fi;
210 push_pbox_explicit("punct.make_comma",u);
211 endgroup;
212 enddef;
213
214 vardef punct.make_revcomma(expr cpos,cang) =
215   begingroup
216     save x,y,t,u,xsp;
217     numeric x[],y[];
218     transform t,u;
219     xsp:=sp;
220     sp:=1;
221     t:=tsu_rescale_xform;
222     sp:=xsp;

```

```

223
224   x1=0.8[x2,x4];
225   (x2+x4)/2=x3=0;
226   (x2-x4)=0.45*(y3-y1)=tsu_punct_size;
227   x5=x3;
228
229   y2=y4=0.32[y3,y1]=0;
230   y5-y4=0.73*(y4-y3);
231
232   push_stroke(z1{curl 0.2}..tension 1.2..z2..z3..z4{dir 280}..
233     z5.{curl 0.2}(point 0.8 of (z1{curl 0.2}..tension 1.2..
234     z2..z3..z4{dir 280})),
235     (2,2)-(2,2)-(2,2)-(2,2)-(2,2)-(2,2)-(2,2));
236   replace_strokep(0)((point 4.2 of oldp)-oldp);
237   (0,0) transformed u=llcorner get_strokep(0);
238   (1,0) transformed u=lrcorner get_strokep(0);
239   (0,1) transformed u=ulcorner get_strokep(0);
240   replace_strokep(0)(reverse (oldp rotated -6 reflectedabout(up,down)));
241   replace_strokep(0)(oldp rotated (cang-6) shifted (cpos transformed t)
242     transformed inverse t);
243   u:=u scaled 1.3 rotated (cang-6) shifted (cpo transformed t)
244     transformed inverse t;
245   set_botip(0,1,0);
246   set_bobrush(0,brpunct);
247
248   if tsu_brush_max.brpunct>=0.3:
249     set_bosize(0,40);
250     push_lcblob(get_strokep(0)-cycle);
251   else:
252     replace_strokep(0)(subpath (0.8,6) of oldp);
253     set_bosize(0,80);
254   fi;
255   push_pbox_explicit("punct.make_revcomma",u);
256 endgroup;
257 endif;

```

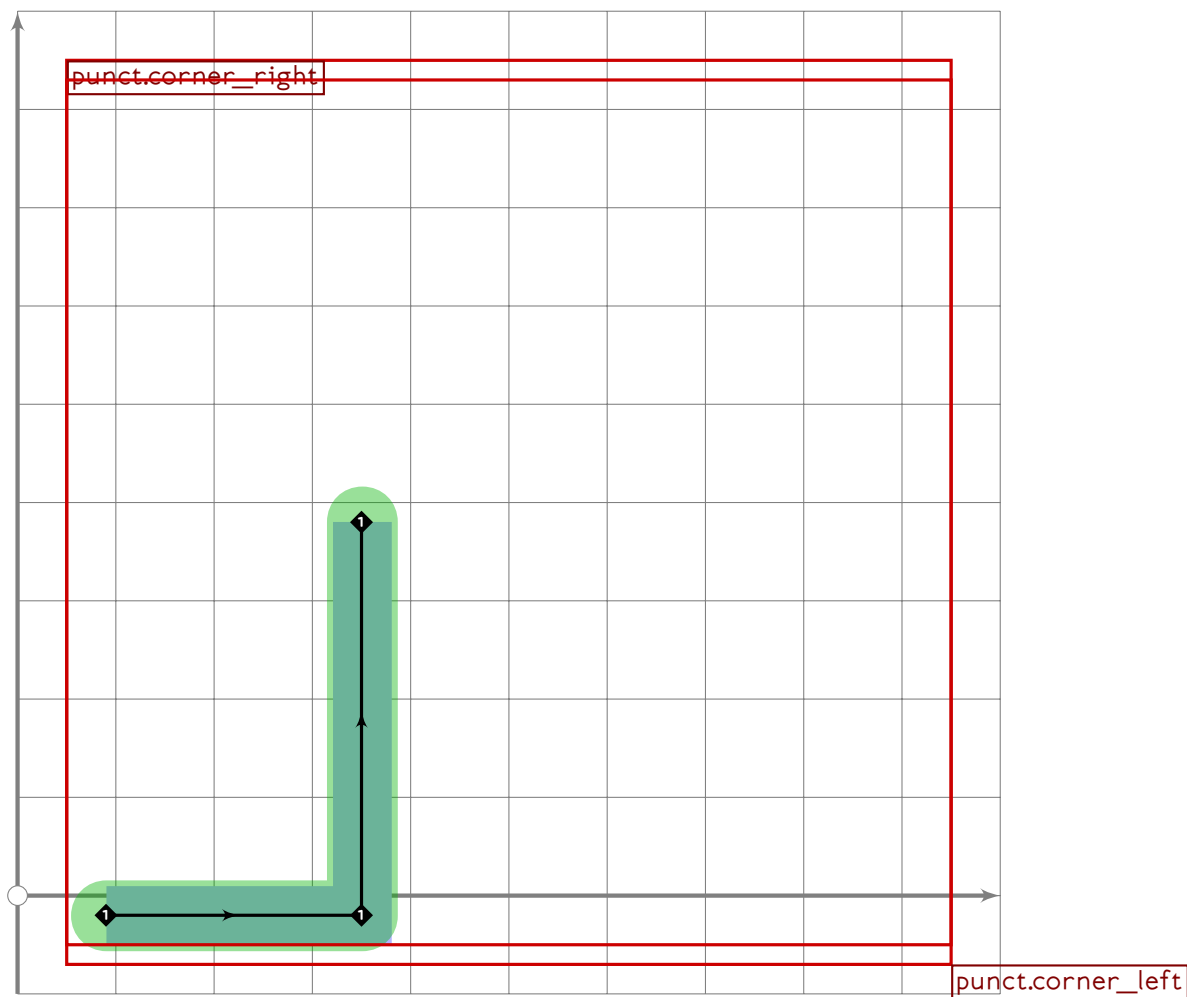


```

258
259 vardef punct.corner_left =
260   push_pbox__toexpand("punct.corner_left");
261   push_stroke((910,800)-(650,800)-(650,400),(2,2)-(2,2)-(2,2));
262   set_bobrush(0,brpunct);
263   set_bosize(0,120);
264   set_botip(0,1,1);
265   expand_pbox;
266 enddef;

```

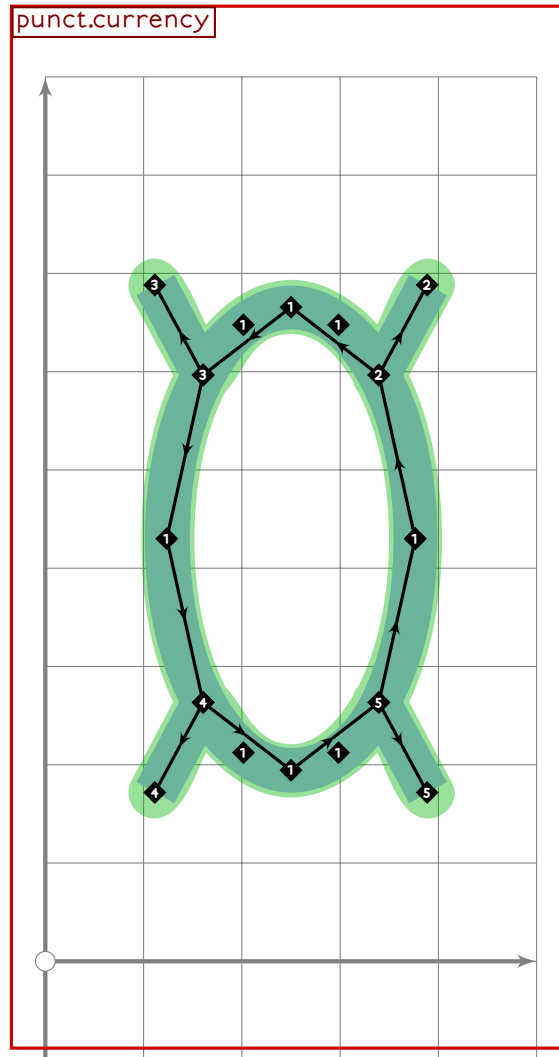
U+300D
tsuku.uni300D



```

267 vardef punct.corner_right =
268   push_pbox_toexpand("punct.corner_right");
269   tsu_xform(identity rotatedaround (centre_pt,180))
270     (punct.corner_left);
271   expand_pbox;
272 enddef;

```



```

273
274 vardef punct.currency =
275   push_pbox_toexpand("punct.currency");
276
277   push_stroke(fullcircle scaled (4*tsu_punct_size) shifted centre_pt,
278     (2,2)-(2,2)-(2,2)-(2,2)-cycle);
279   set_bobrush(0,brpunct);
280   set_bosize(0,90);
281
282   push_stroke(((1,0)-(1.55,0)) rotated 45
283     scaled (2*tsu_punct_size) shifted centre_pt,
284     (2,2)-(2,2));
285   set_bobrush(0,brpunct);
286   set_bosize(0,90);
287
288   push_stroke(((1,0)-(1.55,0)) rotated 135
289     scaled (2*tsu_punct_size) shifted centre_pt,
290     (2,2)-(2,2));
291   set_bobrush(0,brpunct);

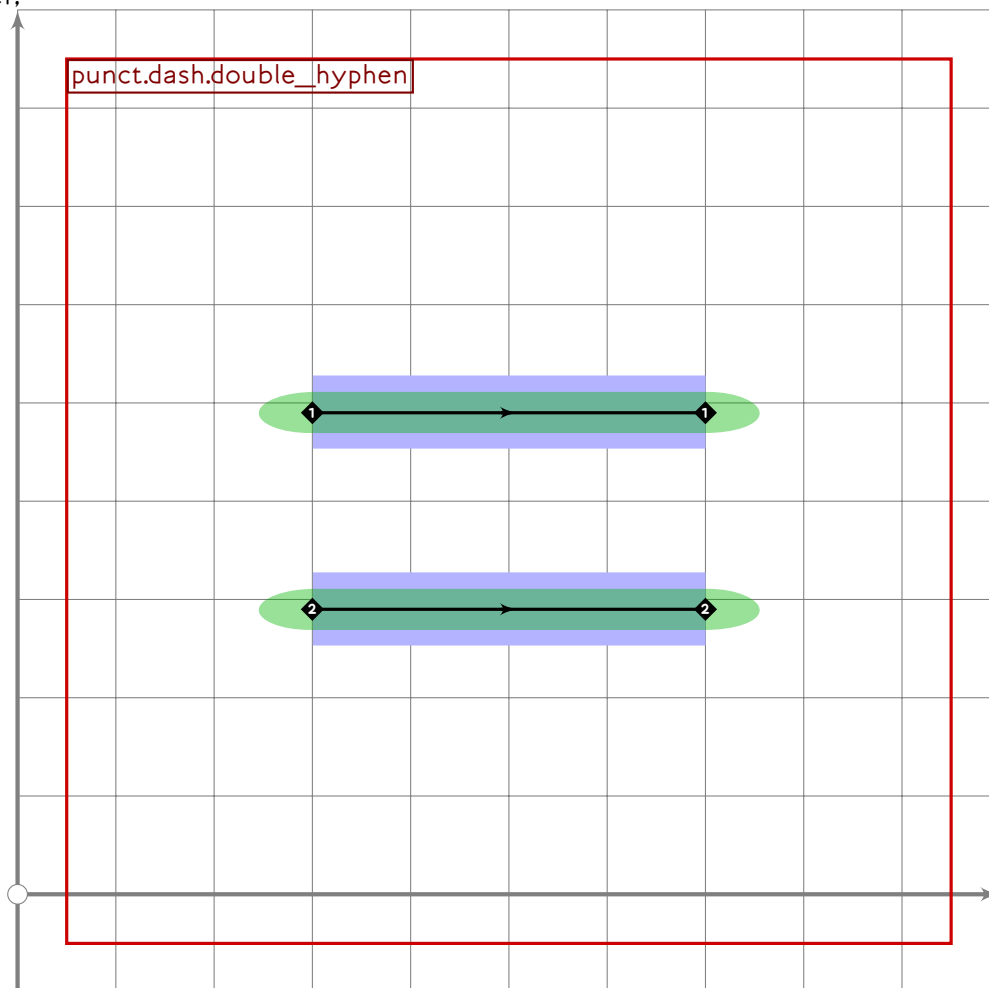
```

U+30A0
tsuku.uni30A0

```

292 set_bosize(0,90);
293
294 push_stroke(((1,0)-(1.55,0)) rotated 225
295     scaled (2*tsu_punct_size) shifted centre_pt,
296     (2,2)-(2,2));
297 set_bobrush(0,brpunct);
298 set_bosize(0,90);
299
300 push_stroke(((1,0)-(1.55,0)) rotated 315
301     scaled (2*tsu_punct_size) shifted centre_pt,
302     (2,2)-(2,2));
303 set_bobrush(0,brpunct);
304 set_bosize(0,90);
305 expand_pbox;
306 endif;

```



```

307
308 vardef punct.dash.double_hyphen =
309     push_pbox_toexpand("punct.dash.double_hyphen");
310
311     (z1+z4)/2=centre_pt;
312     x2-x1=400;

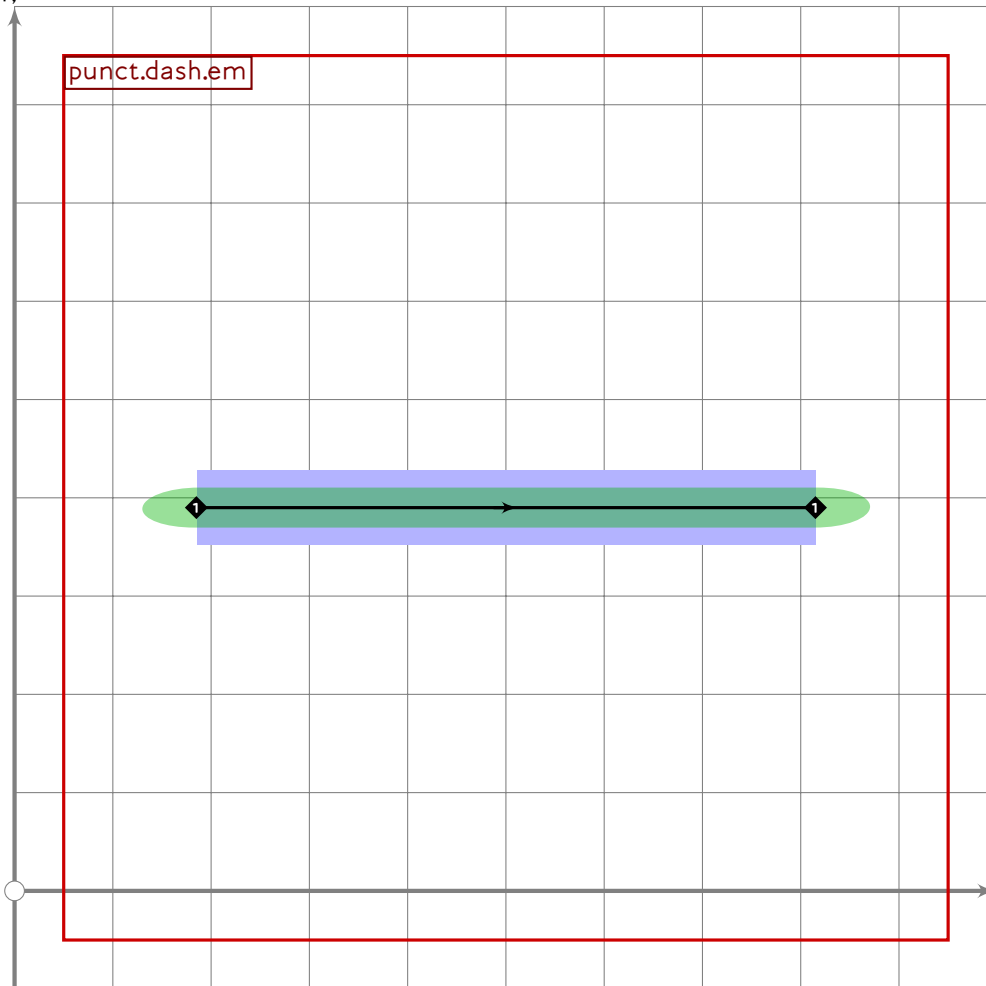
```

PUNC

```

313 y1-y3=200;
314 y1=y2;
315 y3=y4;
316 x1=x3;
317 x2=x4;
318
319 push_stroke(z1-z2,(2,2)-(2,2));
320 % set_bobrush(0,brpunct);
321 push_stroke(z3-z4,(2,2)-(2,2));
322 % set_bobrush(0,brpunct);
323
324 expand_pbox;
325 enddef;

```



```

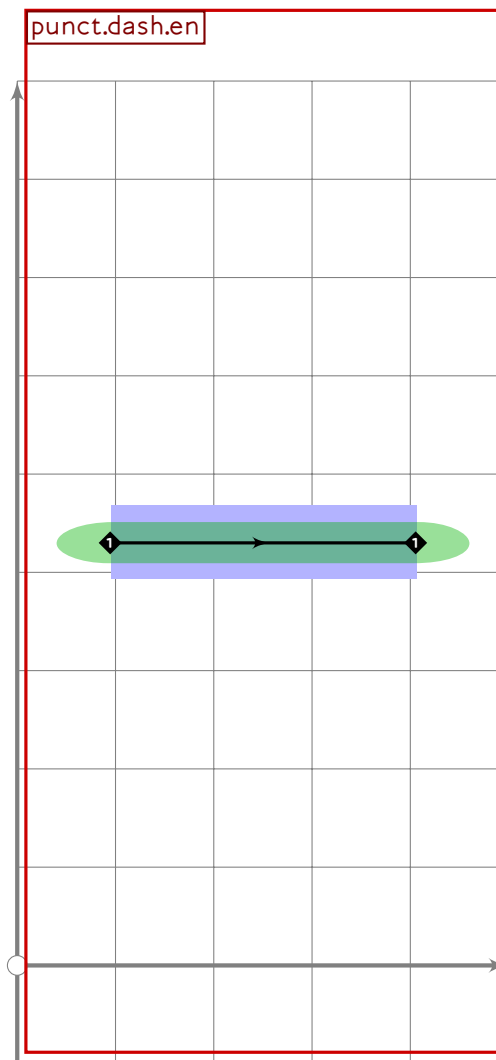
326
327 vardef punct.dash.em =
328   push_pbox_toexpand("punct.dash.em");
329   (z1+z2)/2=centre_pt;
330   x2-x1=if is_proportional: 750 else: 630 fi;
331   y1=y2;
332   push_stroke(z1-z2,(2,2)-(2,2));
333   % set_bobrush(0,brpunct);

```

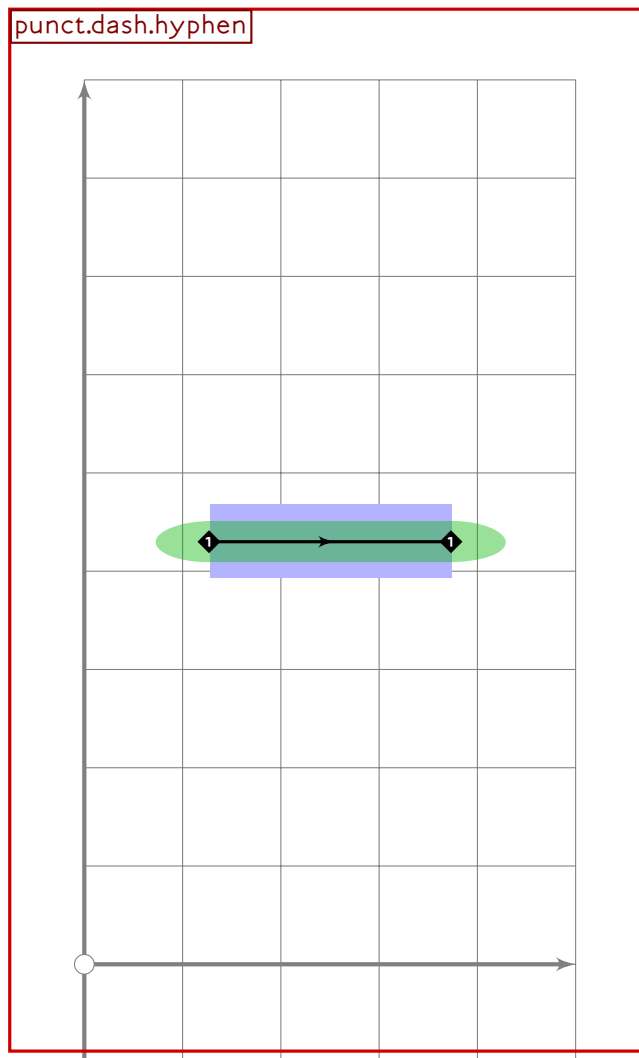
U+2012

tsuku.figuredash

```
334 expand_pbox;  
335 endif;
```



```
336  
337 vardef punct.dash.en =  
338   push_pbox_toexpand("punct.dash.en");  
339   (z1+z2)/2=centre_pt;  
340   x2-x1=580;  
341   y1=y2;  
342   push_stroke(z1-z2,(2,2)-(2,2));  
343   % set_bobrush(0,brpunct);  
344   expand_pbox;  
345 endif;
```

```

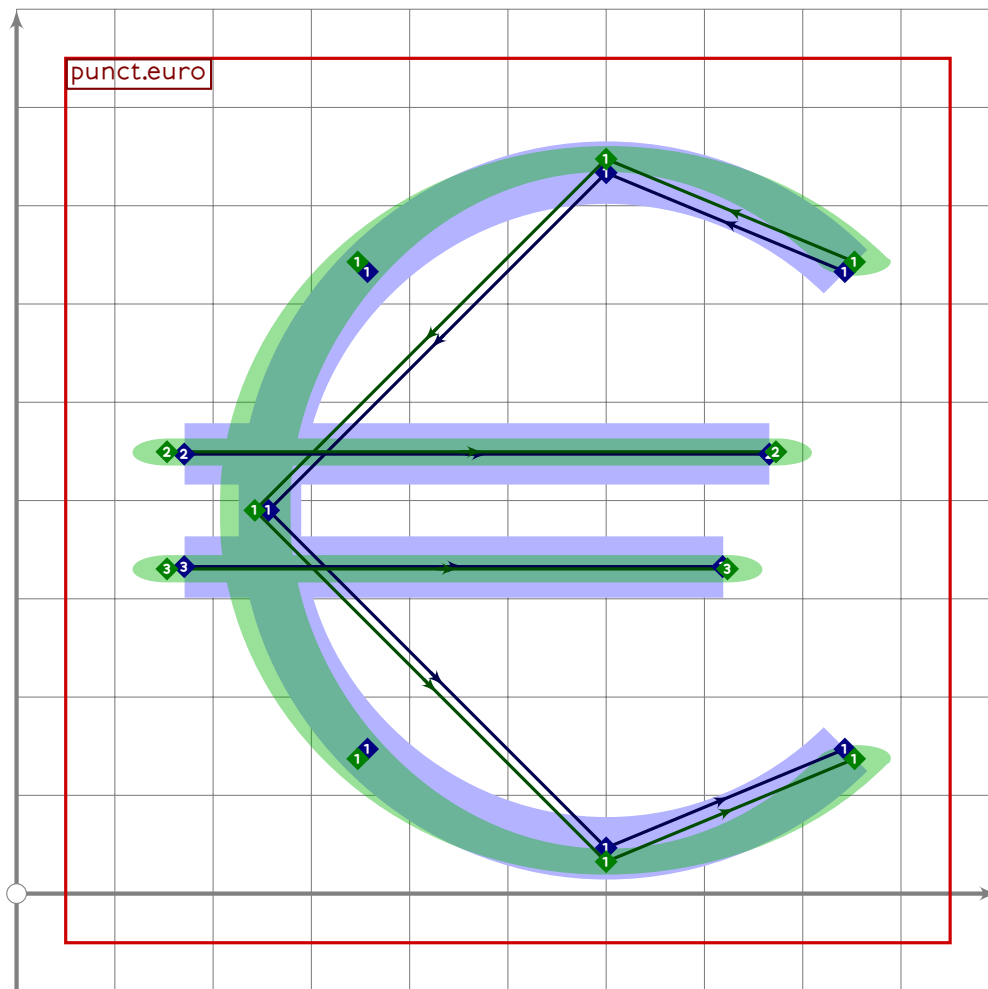
346
347 vardef punct.dash.hyphen =
348   push_pbox_toexpand("punct.dash.hyphen");
349   (z1+z2)/2=centre_pt;
350   x2-x1=340;
351   y1=y2;
352   push_stroke(z1-z2,(2,2)-(2,2));
353   % set_bobrush(0,brpunct);
354   expand_pbox;
355 enddef;
356
357 vardef punct.dash.long =
358   push_pbox_toexpand("punct.dash.long");
359   (z1+z2)/2=centre_pt;
360   x2-x1=340;
361   y1=y2;
362   push_stroke(z1-z2,(2,2)-(2,2));
363   % set_bobrush(0,brpunct);
364   expand_pbox;

```

```

365 endif;
366
367 vardef punct.dividedby(expr t) =
368   push_stroke(((1,0)-(1,0)) transformed t,(2,2)-(2,2));
369   set_bobrush(0,brpunct);
370   set_bosize(0,90);
371
372   push_lcblob(fullcircle scaled (0.65*tsu_punct_size/xxpart t)
373     shifted (0,0.9) transformed t);
374   push_lcblob(fullcircle scaled (0.65*tsu_punct_size/xxpart t)
375     shifted (0,-0.9) transformed t);
376
377   push_pbox_explicit("punct.dividedby",
378     identity shifted (-0.5,-0.5) scaled 2.4 transformed t);
379 endif;
380
381 vardef punct.equals(expr t) =
382   push_stroke(((1,0.667)-(1,0.667)) transformed t,(2,2)-(2,2));
383   set_bobrush(0,brpunct);
384   set_bosize(0,90);
385
386   push_stroke(((1,-0.667)-(1,-0.667)) transformed t,(2,2)-(2,2));
387   set_bobrush(0,brpunct);
388   set_bosize(0,90);
389
390   push_pbox_explicit("punct.equals",
391     identity shifted (-0.5,-0.5) xyscaled (2.4,1.8) transformed t);
392 endif;

```



```

393
394 vardef punct.euro =
395   push_pbox_toexpand("punct.euro");
396
397   push_stroke((subpath (0.5,3.5) of ((1,0)..(0,1)..(-1,0)..(0,-1)..cycle))
398     scaled ((latin_wide_high_r-latin_wide_low_r)/2)
399     shifted (centre_pt+(100,0)),
400     (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
401   set_bosize(0,90);
402
403   push_stroke((( -1.25,0.1667)-
404     (((0,0.1667)-(1,0.1667)) intersectionpoint ((0.707,0.707)-(0,-1))))
405     scaled ((latin_wide_high_r-latin_wide_low_r)/2)
406     shifted (centre_pt+(100,0)),
407     (1.6,1.6)-(1.6,1.6));
408   set_bosize(0,90);
409
410   push_stroke((( -1.25,-0.1667)-
411     (((0,-0.1667)-(1,-0.1667)) intersectionpoint ((0.707,0.707)-(0,-1))))
412     scaled ((latin_wide_high_r-latin_wide_low_r)/2)
413     shifted (centre_pt+(100,0)),

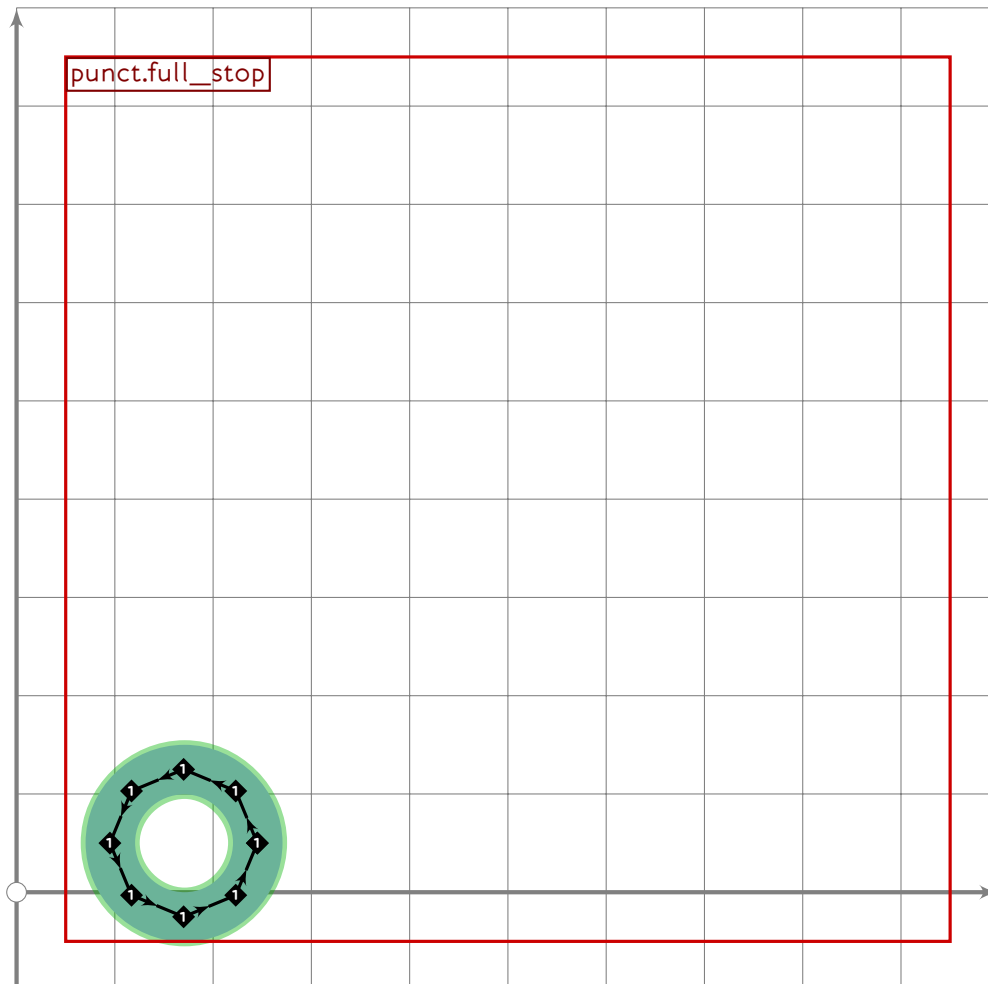
```

U+3002
tsuku.uni3002

```

414     (1.6,1.6)-(1.6,1.6));
415     set_bosize(0,90);
416     expand_pbox;
417 endif;
418

```



```

419 % this is *ideographic* full stop, not Latin period
420 vardef punct.full_stop =
421   push_pbox_toexpand("punct.full_stop");
422
423   if tsu_brush_max.brpunct*100>=tsu_punct_size:
424     push_lcblob(fullcircle
425       xscaled (1.5*tsu_punct_size+tsu_brush_max.brpunct*100)
426       yscaled (1.5*tsu_punct_size+tsu_brush_max.brpunct*100
427         *tsu_brush_shape.brpunct)
428       rotated tsu_brush_angle.brpunct
429       shifted (170,50));
430   else:
431     push_stroke(fullcircle scaled (1.5*tsu_punct_size) shifted (170,50),
432       (2,2)-(2,2)-(2,2)-(2,2)-cycle);
433     set_bobrush(0,brpunct);
434   fi;

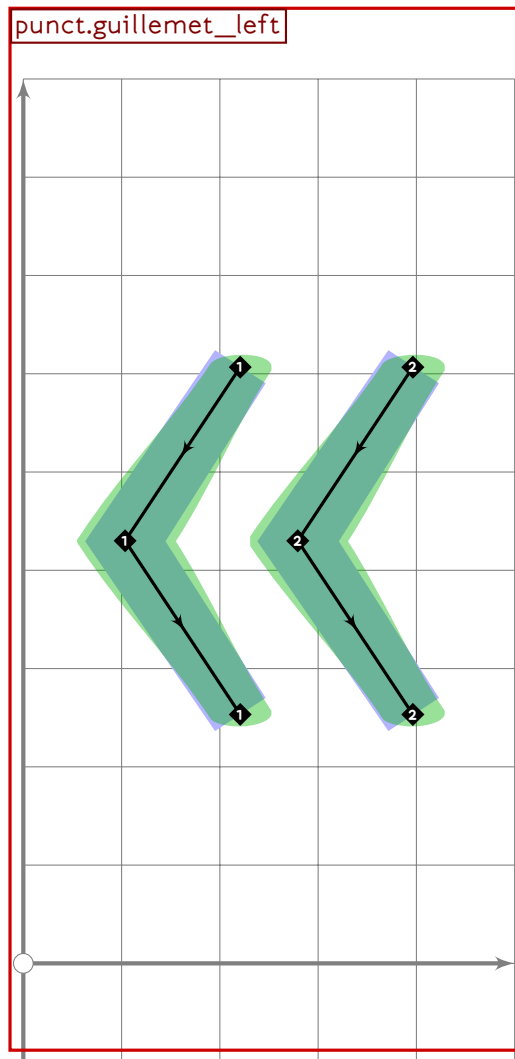
```

PUNC

```

435 expand_pbox;
436 enddef;
437
438 vardef punct.greater_than(expr t) =
439   push_stroke(((−1,1)−(1,0)−(−1,−1)) transformed t,(2,2)−(2,2)−(2,2));
440   set_bobrush(0,brpunct);
441   set_bosize(0,90);
442   set_botip(0,1,1);
443
444   push_pbox_explicit("punct.greater_than",
445     identity shifted (−0.5,−0.5) scaled 2.4 transformed t);
446 enddef;

```



```

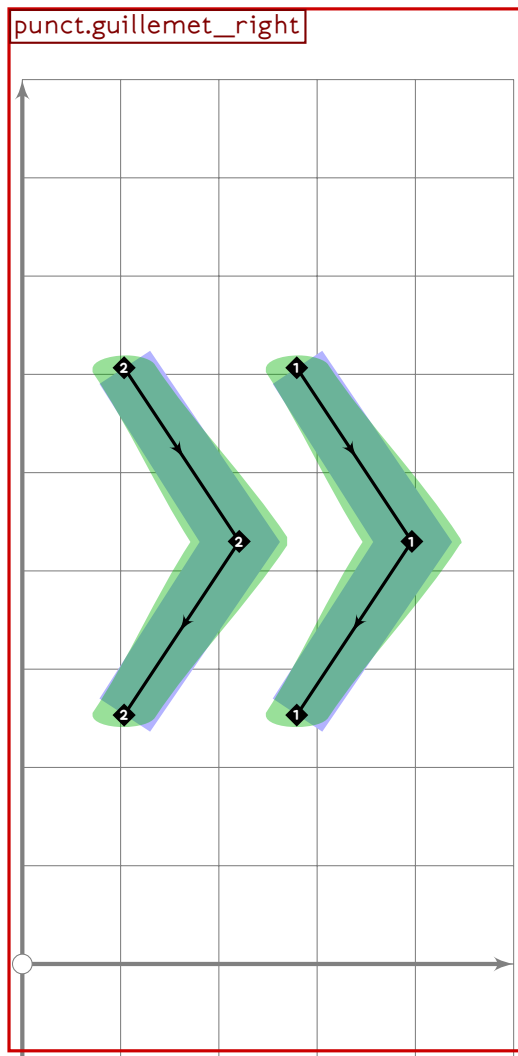
447
448 vardef punct.guillemet_left =
449   push_pbox_toexpand("punct.guillemet_left");
450
451   push_stroke(((−0.5,1.5)−(−2.5,0)−(−0.5,−1.5))
452     scaled tsu_punct_size shifted centre_pt,
453     (1.5,1.5)−(2,2)−(1.5,1.5));

```

U+00BB

tsuku.guillemotright

```
454 set_bosize(0,90);
455 set_botip(0,1,1);
456
457 push_stroke(((2.5,1.5)-(0.5,0)-(2.5,-1.5))
458     scaled tsu_punct_size shifted centre_pt,
459     (1.5,1.5)-(2,2)-(1.5,1.5));
460 set_bosize(0,90);
461 set_botip(0,1,1);
462 expand_pbox;
463 enddef;
```

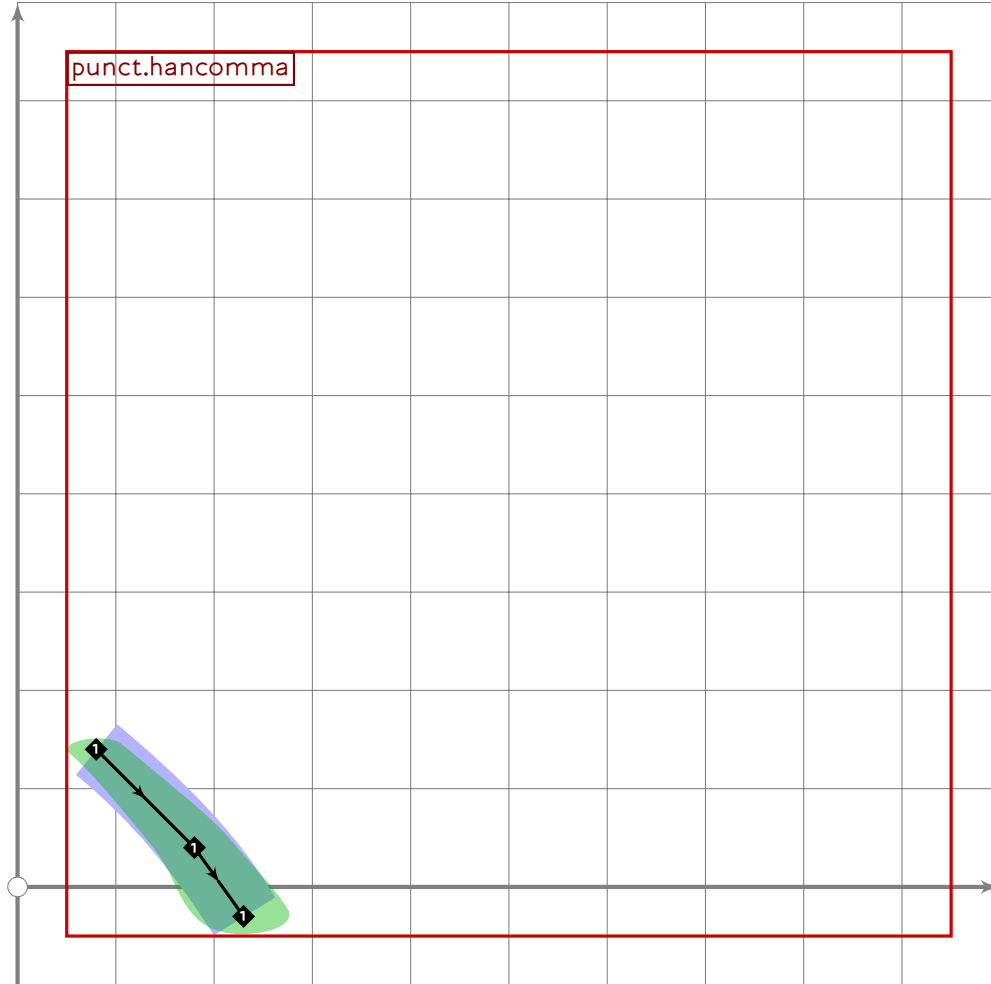


```
464
465 vardef punct.guillemet_right =
466     push_pbox_toexpand("punct.guillemet_right");
467
468     push_stroke(((0.5,1.5)-(2.5,0)-(0.5,-1.5))
469         scaled tsu_punct_size shifted centre_pt,
470         (1.5,1.5)-(2,2)-(1.5,1.5));
471     set_bosize(0,90);
472     set_botip(0,1,1);
```

```

473
474 push_stroke(((−2.5,1.5)−(−0.5,0)−(−2.5,−1.5))
475     scaled tsu_punct_size shifted centre_pt,
476     (1.5,1.5)−(2,2)−(1.5,1.5));
477 set_bosize(0,90);
478 set_botip(0,1,1);
479 expand_pbox;
480 enddef;

```



```

481
482 vardef punct.hancomma =
483   push_pbox_toexpand("punct.hancomma");
484   push_stroke((80,140)..(180,40)..(230,−30),(1.3,1.3)..(1.6,1.6)..(1.8,1.8));
485   expand_pbox;
486 enddef;
487
488 vardef punct.hminus(expr t) =
489   push_stroke(((−1,0)−(1,0)) transformed t,(2,2)−(2,2));
490   % set_bobrush(0,brpunct);
491
492   push_pbox_explicit("punct.hminus";
493     identity shifted (−0.5,−0.5) xyscaled (2.4,0.6) transformed t);

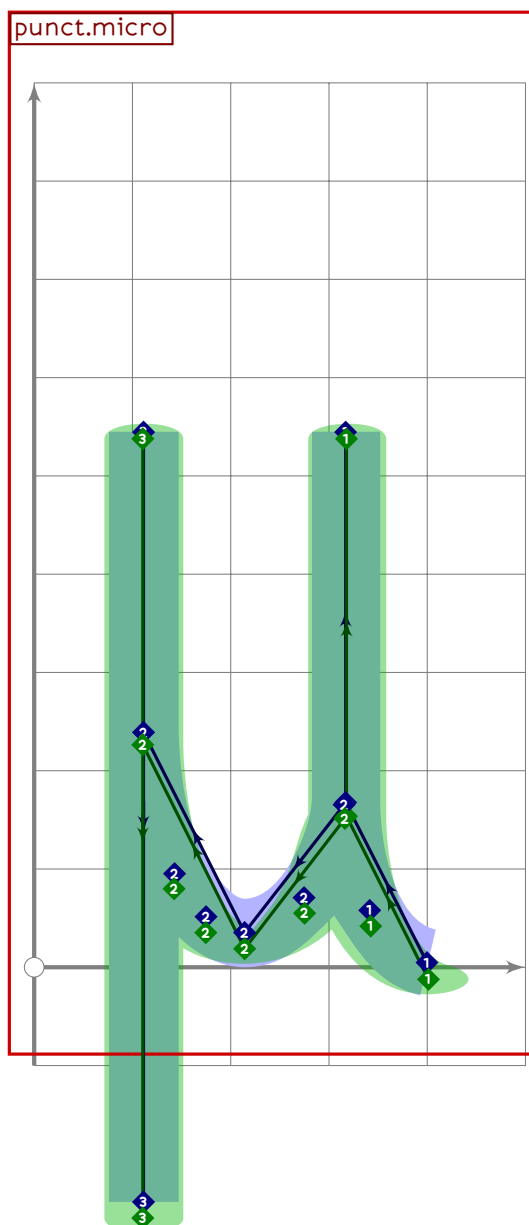
```

U+00B5
tsuku.mu

```

494 endif;
495
496 vardef punct.less_than(expr t) =
497   push_stroke(((1,1)-(-1,0)-(-1,-1)) transformed t,(2,2)-(2,2)-(2,2));
498   set_bobrush(0,brpunct);
499   set_bosize(0,90);
500   set_botip(0,1,1);
501
502   push_pbox_explicit("punct.less_than",
503     identity shifted (-0.5,-0.5) scaled 24 transformed t);
504 endif;
505

```



PUNC

```

506 % in the future, this will probably become greek.lowmu
507 vardef punct.micro =
508   push_pbox_toexpand("punct.micro");

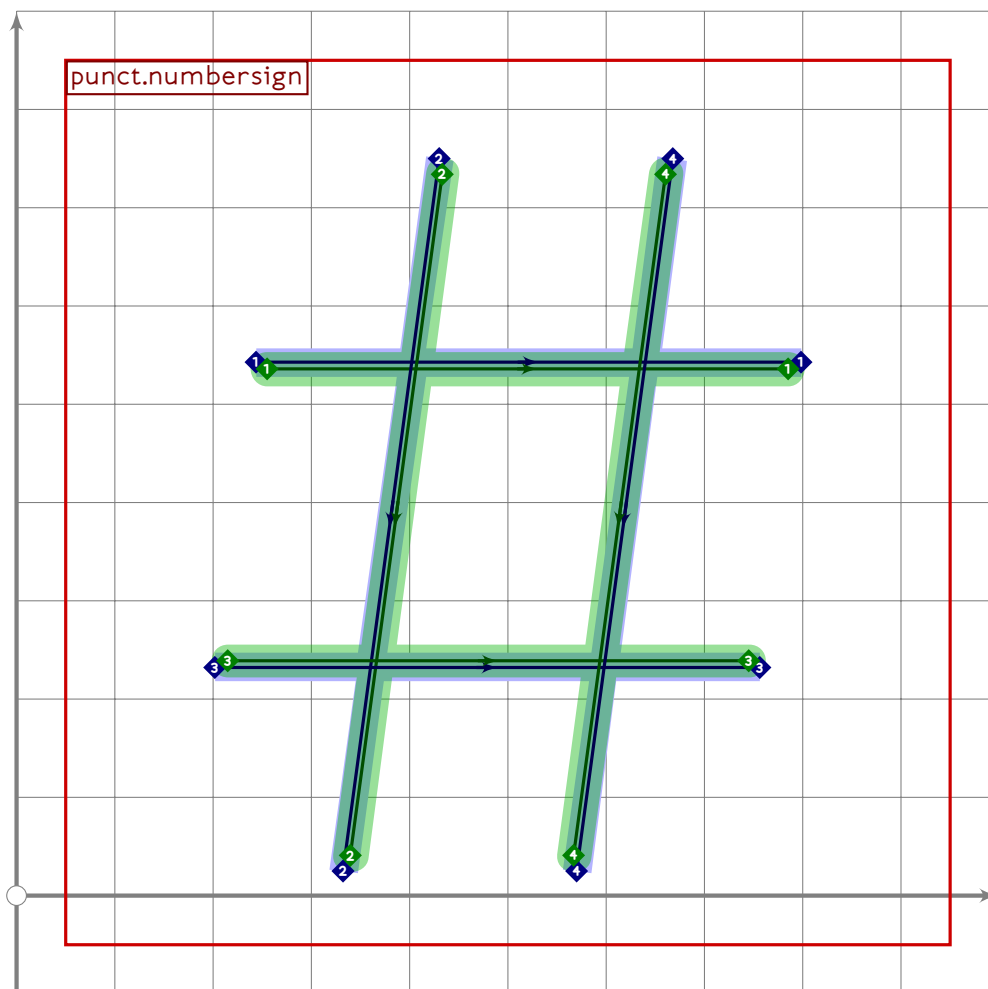
```



```

509
510 x1-x2=y2-y1;
511 (x2+x6)/2=450;
512 (x2-x6)=(y3-y1)*0.75;
513 x3=x2=x4;
514 x5=0.5[x4,x6];
515 x7=x8=x6;
516
517 y1=(-0.06)[y5,y3];
518 y2=0.26[y5,y3];
519 y3=y7=latin_wide_xheight_v;
520 y4=0.73[y3,y5];
521 y5=latin_wide_low_h;
522 y6=0.60[y3,y5];
523 y8=latin_wide_desc_v;
524
525 push_stroke(z1{dir 173}..{up}z2-z3,(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
526 push_stroke(subpath (0.03,2) of (z4..z5{left}..z6{dir 93}),
527   (1.6,1.6)-(1.6,1.6)-(1.6,1.6));
528 push_stroke(z7-z8,(1.6,1.6)-(1.6,1.6));
529 expand_pbox;
530 enddef;
531
532 vardef punct.otsign(expr t) =
533   push_stroke(((1,0)-(1,0)-(1,1)) transformed t,(2,2)-(2,2)-(2,2));
534   set_bobrush(0,brpunct);
535   set_bosize(0,90);
536   set_botip(0,1,1);
537
538   push_pbox_explicit("punct.otsign",
539     identity shifted (-0.5,-0.5) scaled 24 transformed t);
540 enddef;

```



```

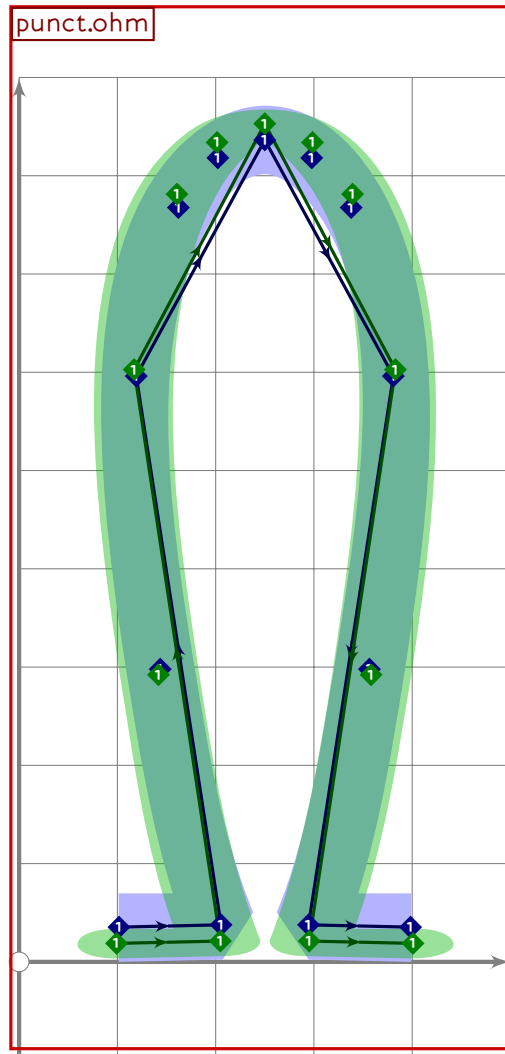
541
542 vardef punct.numbersign =
543   push_pbox_toexpand("punct.numbersign");
544
545   (x1+x2)/2=500;
546   (x2-x1)=0.9*(y2-y1);
547   x3=0.15[x1,x2];
548
549   y1=latin_wide_low_v;
550   y2=y3=latin_wide_high_v;
551
552   transform xf_num;
553   (0,0) transformed xf_num = z1;
554   (3.5,3.5) transformed xf_num = z2;
555   (0,3.5) transformed xf_num = z3;
556
557   push_stroke(((0,2.5)-(3.5,2.5)) transformed xf_num,(1.6,1.6)-(1.6,1.6));
558   set_bobrush(0,brpunct);
559   set_bosize(0,85);
560   push_stroke(((1,3.5)-(1,0)) transformed xf_num,(1.6,1.6)-(1.6,1.6));
561   set_bobrush(0,brpunct);

```

```

562 set_bosize(0,85);
563 push_stroke(((0,1)-(3.5,1)) transformed xf_num,(1.6,1.6)-(1.6,1.6));
564 set_bobrush(0,brpunct);
565 set_bosize(0,85);
566 push_stroke(((2.5,3.5)-(2.5,0)) transformed xf_num,(1.6,1.6)-(1.6,1.6));
567 set_bobrush(0,brpunct);
568 set_bosize(0,85);
569 expand_pbox;
570 enddef;
571

```



```

572 % in the future, this will probably become greek.upomega
573 vardef punct.ohm =
574   push_pbox_toexpand("punct.ohm");
575
576   (x5+x3)/2=(x6+x2)/2=(x7+x1)/2=x4=500;
577   x2=0.7[x1,x4];
578   x7-x1=0.76*(y4-y1);
579   x5-x3=0.67*(y4-y1);
580

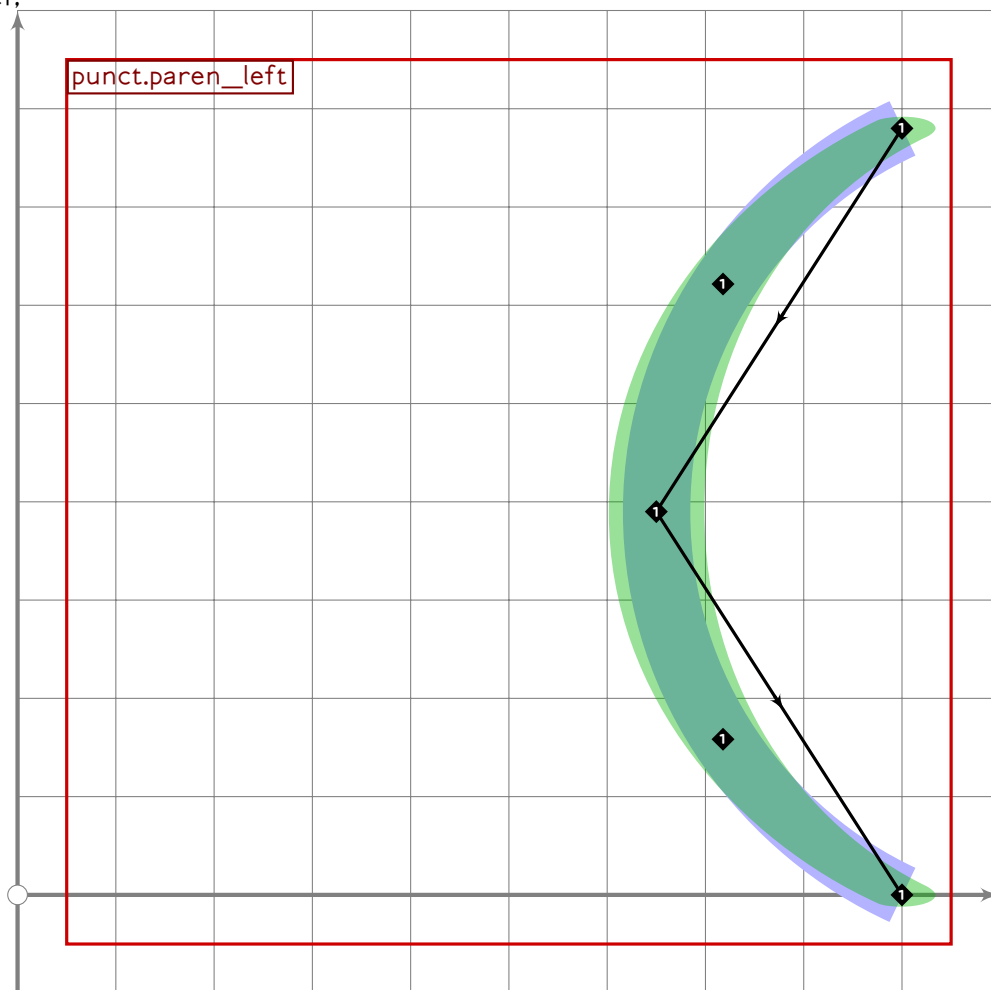
```

U+FF08
tsuku.uniFF08

```

581 y1=y7=latin_wide_low_h;
582 y2=y6=y1+2;
583 y3=y5=0.7[y1,y4];
584 y4=latin_wide_high_r;
585
586 push_stroke(z1-z2..tension 1.5 and 1.3..z3..z4..
587     z5..tension 1.3 and 1.5..z6-z7,
588     (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-
589     (1.6,1.6)-(1.6,1.6));
590 set_botip(0,1,0);
591 set_botip(0,5,0);
592 expand_pbox;
593 enddef;

```

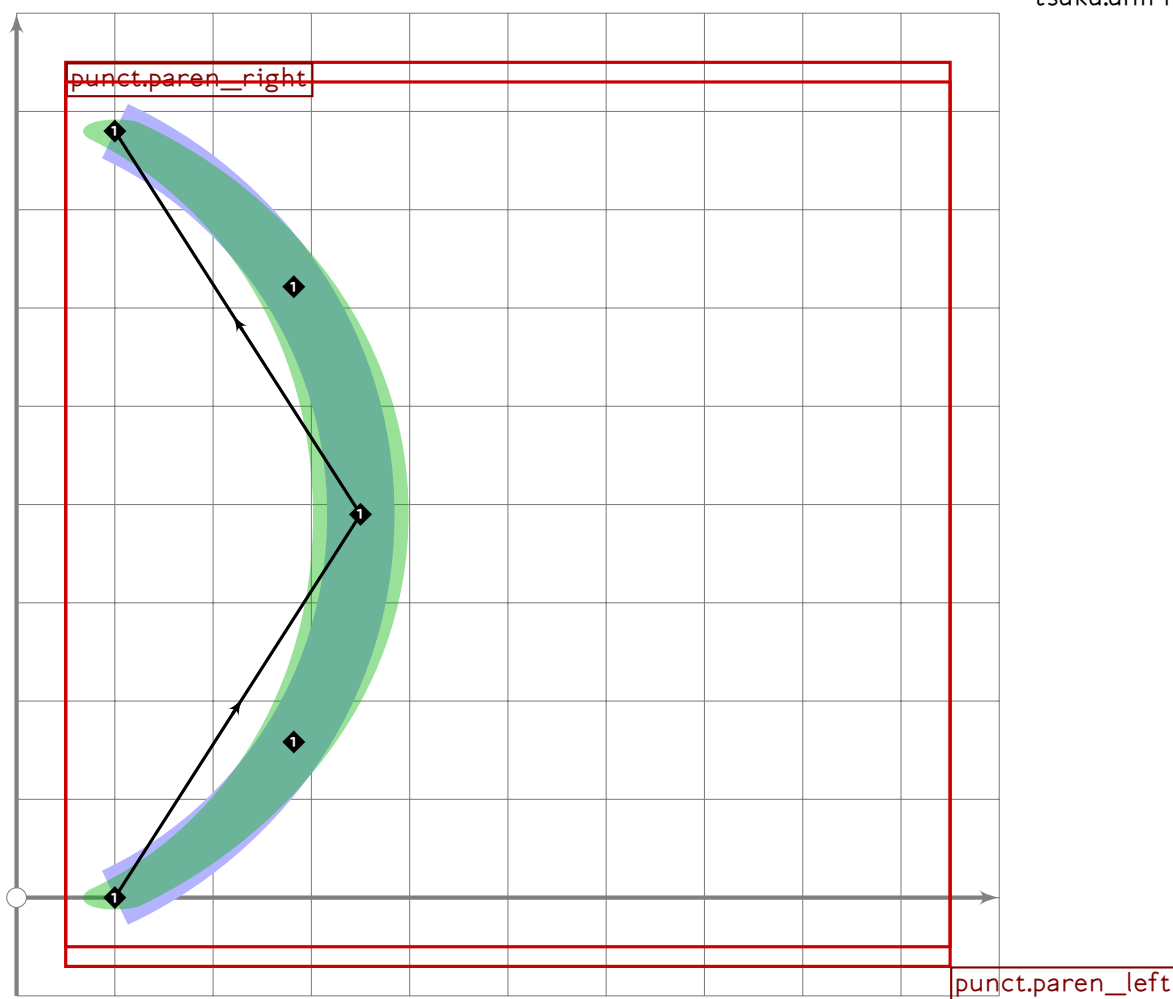


```

594
595 vardef punct.paren_left =
596     push_pbox_toexpand("punct.paren_left");
597     push_stroke((900,780)..(900-2.5*tsu_punct_size,390)..(900,0),
598         (1.5,1.5)-(2,2)-(1.5,1.5));
599     set_bosize(0,90);
600     expand_pbox;
601 enddef;

```

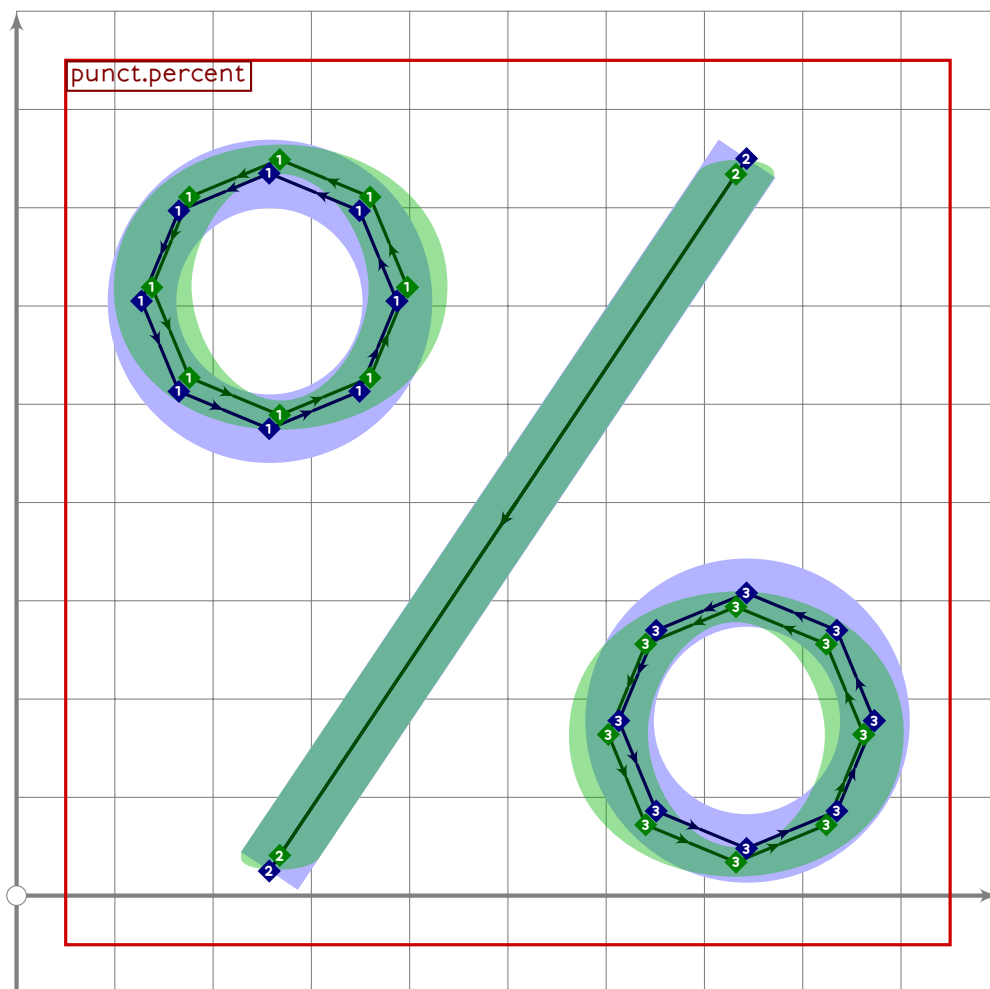
PUNC



```

602
603 vardef punct.paren_right =
604   push_pbox__toexpand("punct.paren_right");
605   tsu_xform(identity rotatedaround (centre_pt,180))
606   (punct.paren_left);
607   expand_pbox;
608 enddef;

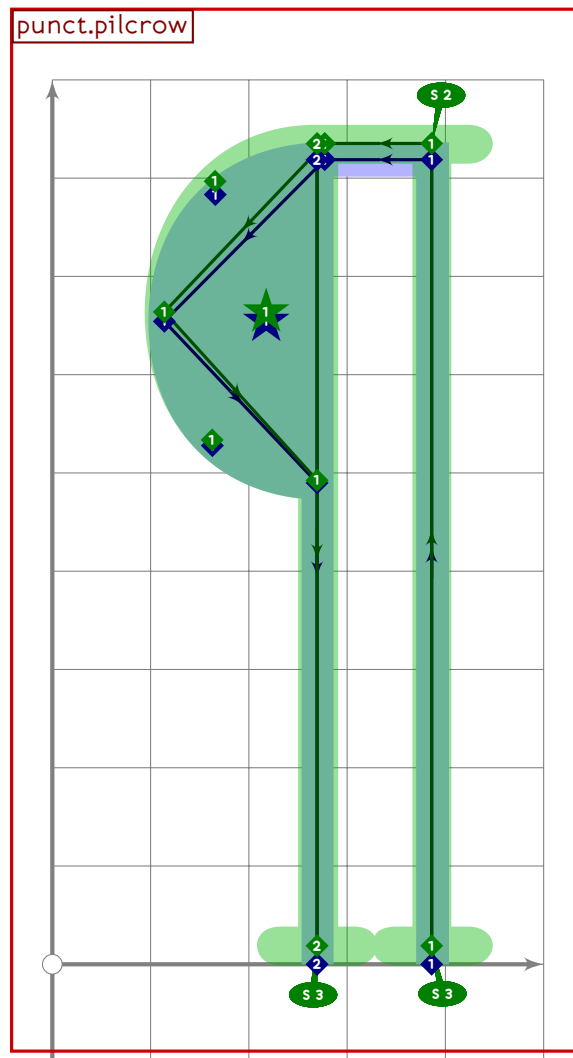
```



```

609
610 vardef punct.percent =
611   push_pbox__toexpand("punct.percent");
612
613   (x1+x2)/2=500;
614   (x1-x2)=0.67(y1-y2);
615
616   y1=latin_wide_high_v;
617   y2=latin_wide_low_v;
618
619   push_stroke(fullcircle scaled 260 shifted (x2,latin_wide_high_r-130),
620     (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-cycle);
621
622   push_stroke((z1-z2)
623     shifted -centre_pt scaled (tsu_punct_size/100) shifted centre_pt,
624     (1.6,1.6)-(1.6,1.6));
625
626   push_stroke(fullcircle scaled 260 shifted (x1,latin_wide_low_r+130),
627     (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-cycle);
628   expand_pbox;
629 enddef;

```



```

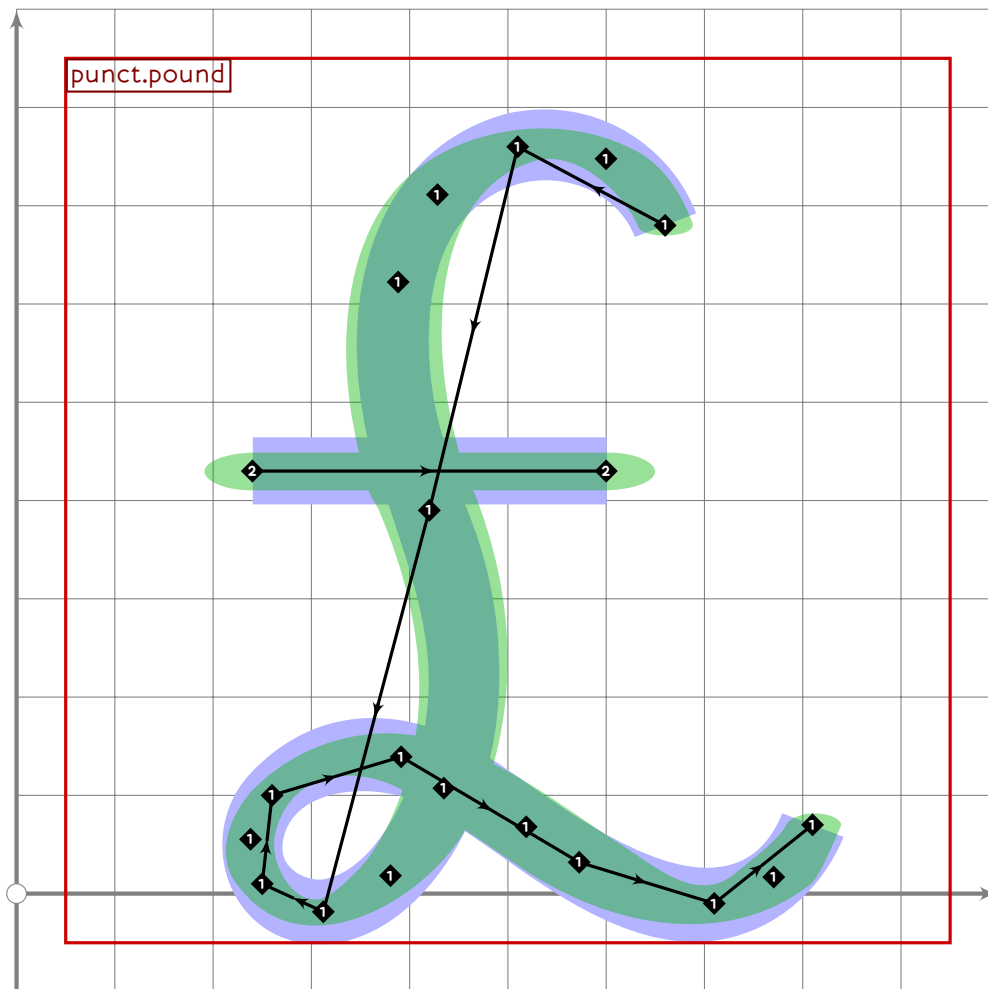
630
631 vardef punct.pilcrow =
632   push_pbox_toexpand("punct.pilcrow");
633
634   x1=x2=x6=710;
635   x3=x5=x1-420*0.4;
636   x4=x1-420;
637
638   y1=latin_wide_low_v;
639   y2=y3=latin_wide_high_h;
640   y4=(y3+y5)/2;
641   y5=y6=vmetric(0.58);
642
643   x7=x8=x9=x1-1.8*tsu_punct_size;
644   y7=y2+50;
645   y8=y4;
646   y9=y1;
647
648   push_stroke(z1-z2-z3{left}..z4..{right}z5-z6,

```

```

649 (2,2)-(2,2)-(2,2)-(2,2)-(2,2)-(2,2));
650 replace_strokeq(0)(subpath (0,xpart (get_strokeq(0) intersectiontimes
651 (z8-z9))) of oldq);
652 replace_strokep(0)(subpath (0,xpart (oldp intersectiontimes
653 (z8-z9))) of oldp);
654 set__bobrush(0,brpunct);
655 set__bosize(0,67);
656 set__botip(0,1);
657 set__boserif(0,0,3);
658 set__boserif(0,1,2);
659
660 push_stroke(((z7-z8) intersectionpoint get_strokep(0))-z9,(2,2)-(2,2));
661 set__bobrush(0,brpunct);
662 set__bosize(0,67);
663 set__boserif(0,1,3);
664
665 if tsu_brush_max.brpunct>=0.3:
666   push_lcblob((subpath (xpart (get_strokep(-1) intersectiontimes (z7-z8)),
667     infinity) of get_strokep(-1))-cycle);
668 fi;
669 expand_pbox;
670 enddef;
671
672 vardef punct.plus(expr t) =
673   push_stroke(((1,0)-(1,0)) transformed t,(2,2)-(2,2));
674   set__bobrush(0,brpunct);
675   push_stroke(((0,1)-(0,1)) transformed t,(2,2)-(2,2));
676   set__bobrush(0,brpunct);
677   push_pbox_explicit("punct.plus",
678     identity shifted (-0.5,-0.5) scaled 2.4 transformed t);
679 enddef;
680
681 vardef punct.plusminus(expr t) =
682   push_stroke((-1,0.25)-(1,0.25)) transformed t,(2,2)-(2,2));
683   set__bobrush(0,brpunct);
684   push_stroke((0,1.25)-(0,-0.75)) transformed t,(2,2)-(2,2));
685   set__bobrush(0,brpunct);
686   push_stroke((-1,-1.25)-(1,-1.25)) transformed t,(2,2)-(2,2));
687   set__bobrush(0,brpunct);
688
689   push_pbox_explicit("punct.plusminus",
690     identity shifted (-0.5,-0.5) xyscaled (2.4,3.2) transformed t);
691 enddef;

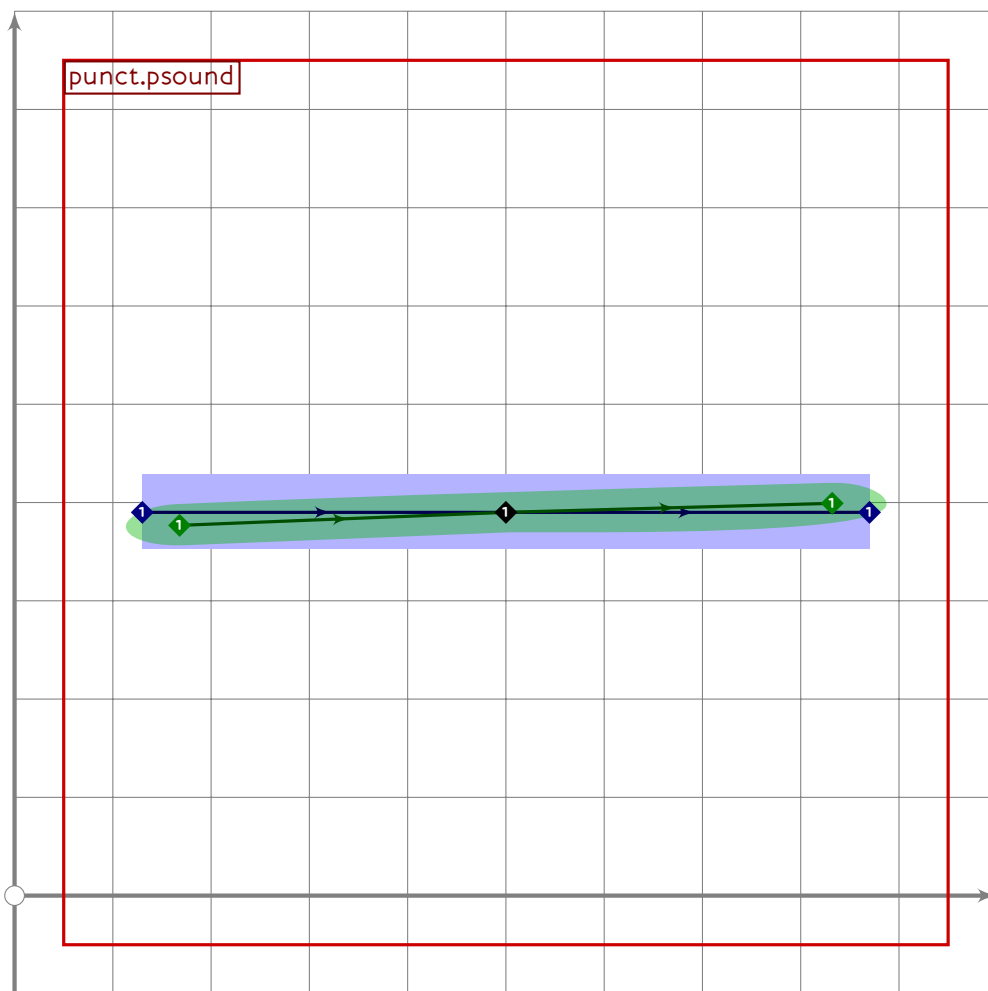
```

```

692
693 vardef punct.pound =
694   push_pbox__toexpand("punct.pound");
695
696   push_stroke((660,680)..(510,760)..(420,390)..tension 11..(250,10)..
697     (260,100)..(710,-10)..(810,70),
698     (1,3,1,3)-(1,7,1,7)-(1,9,1,9)-(1,4,1,4)-(1,2,1,2)-
699     (1,1,1,2)-(2,2)-(2,1,2,1)-(2,2)-(1,3,1,3));
700   replace_strokep(0)(insert_nodes(oldp)(2,8,4,34,7));
701
702   push_stroke((240,430)-(600,430),(2,2)-(2,2));
703   set_bosize(0,90);
704   expand_pbox;
705 enddef;

```

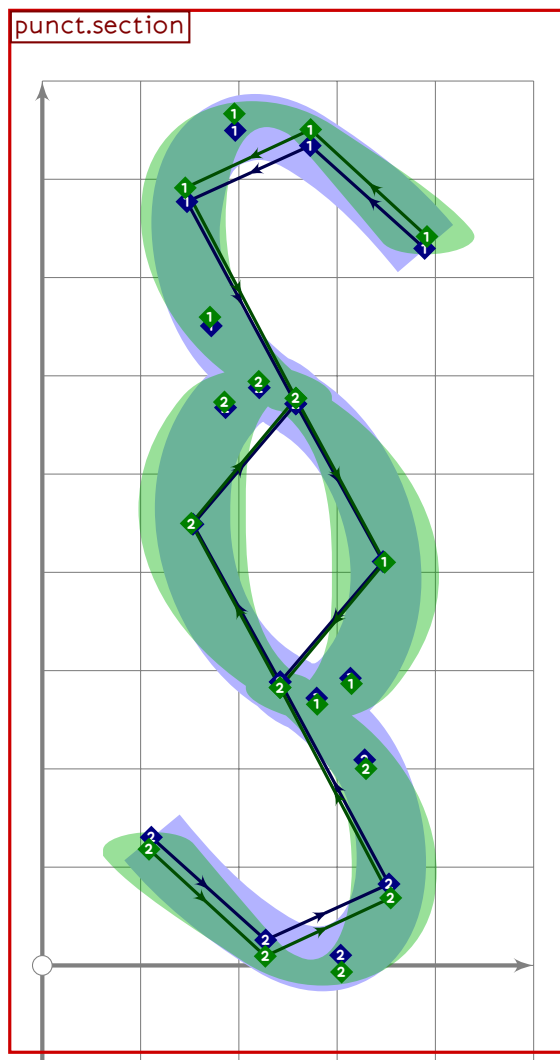


```

706
707 vardef punct.psound =
708   push_pbox__toexpand("punct.psound");
709   push_stroke((130,390-15*mincho)..(500,390)..(870,390+10*mincho),
710     (0.7,3.3)-(2,2)-(0.7,3.3));
711   expand_pbox;
712 enddef;
713
714 vardef punct.make__period(expr cpos) =
715   push_stroke(fullcircle scaled (tsu_punct_size*1.15) shifted cpos,
716     (2,2)-(2,2)-(2,2)-(2,2)-cycle);
717   set_bobrush(0,brpunct);
718
719   if tsu_brush__max.brpunct>=0.3:
720     set_bosize(0,40);
721     push_lcblob(get__strokep(0));
722   else:
723     set_bosize(0,80);
724   fi;
725
726   push_pbox__explicit("punct.make__period",

```

```
727 identity shifted (-0.5,-0.5) scaled (tsu_punct_size*1.5) shifted cpos);
728 endif;
```



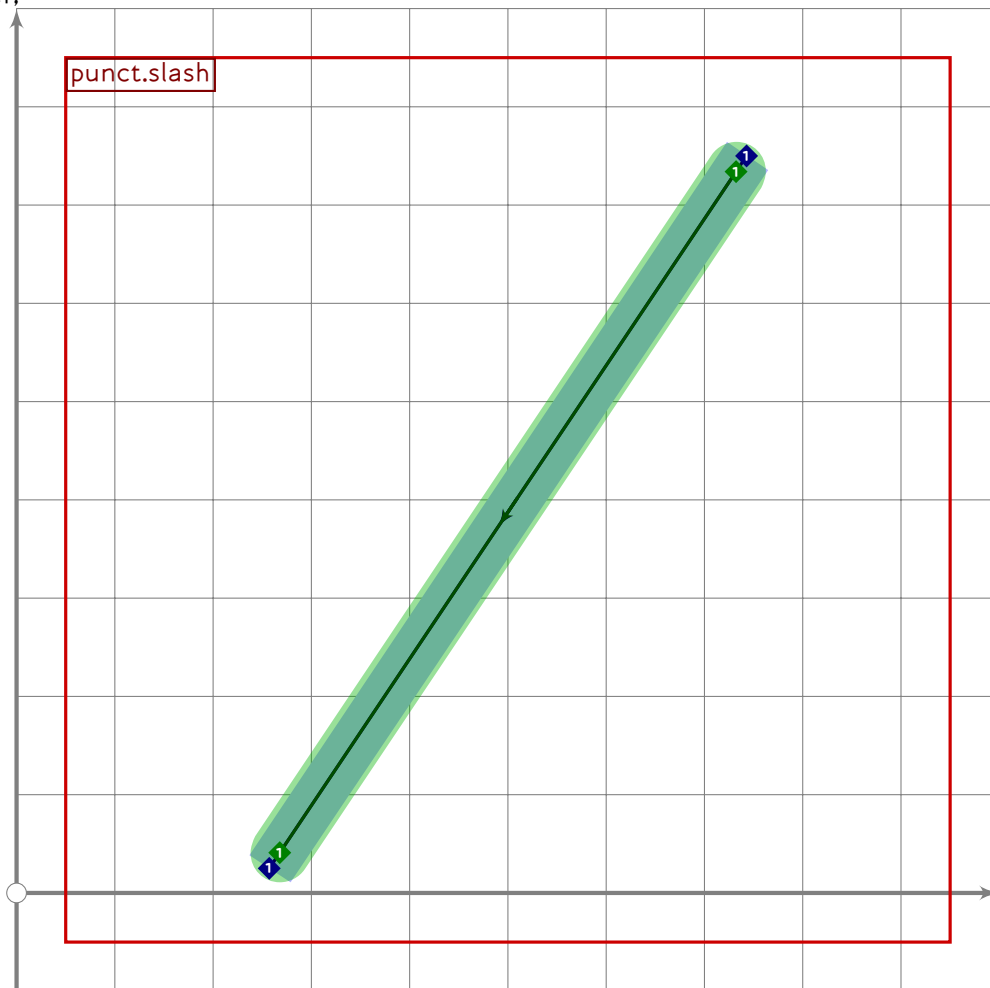
```
729
730 vardef punct.section =
731   push_pbox_toexpand("punct.section");
732
733   (x1+x3)/2=x2=x4=x6=500;
734   x5=0.8[x2,x1];
735   2*(x5-x2)=0.45*(latin_wide_high_r-latin_wide_low_r);
736
737   y1=y3=0.8[ypart centre_pt,latin_wide_high_r];
738   y2=latin_wide_high_r;
739   y4=0.35[ypart centre_pt,latin_wide_high_r];
740   y5=ypart centre_pt;
741   y4-y5=y5-y6;
742
743   push_stroke((z1..z2..z3..z4..z5..z6) rotatedaround (centre_pt,-6),
744     (1.8,1.8)-(1.2,1.2)-(1.7,1.7)-(1.3,1.3)-(2,2)-(1.5,1.5));
745
```

U+FF0F
tsuku.uniFF0F

```

746 push_stroke(get_strokep(0) rotatedaround (centre_pt,180),get_strokeq(0));
747 expand_pbox;
748 enddef;

```

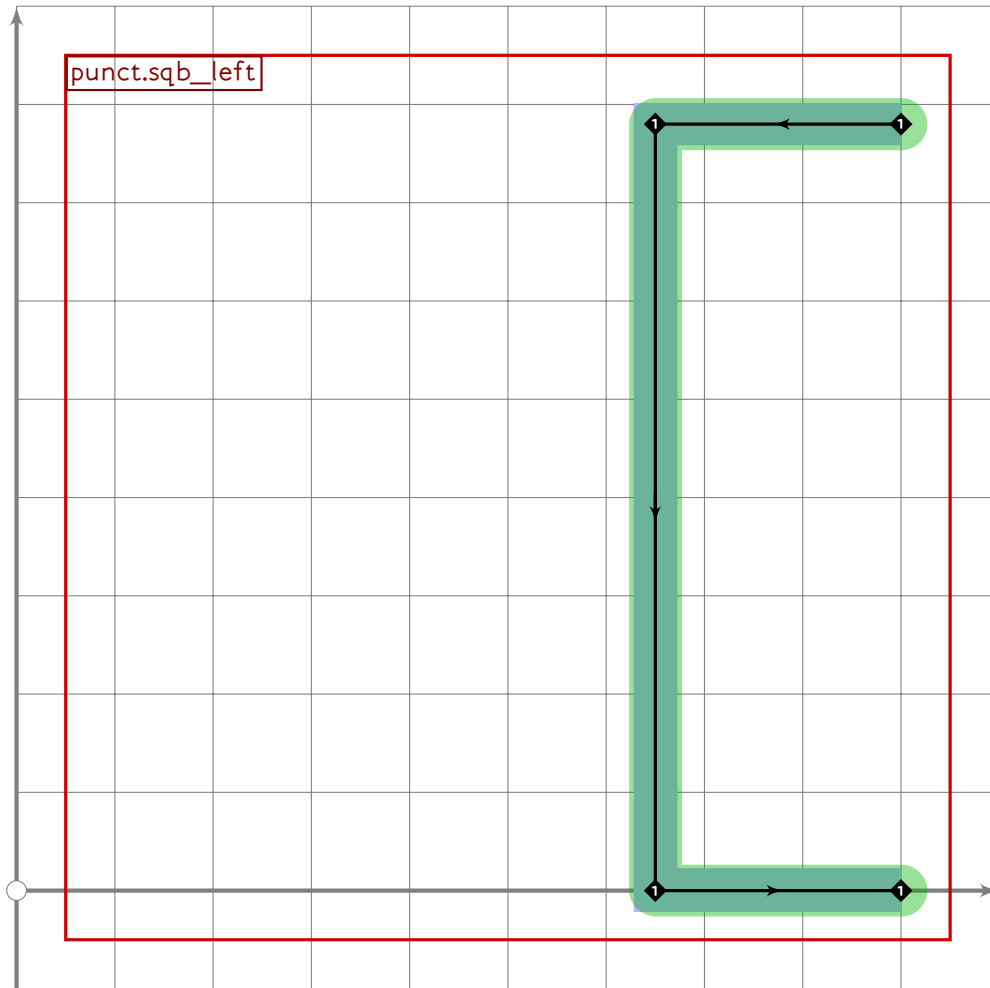


```

749
750 vardef punct.slash =
751   push_pbox_toexpand("punct.slash");
752   (x1+x2)/2=500;
753   (x1-x2)=0.67(y1-y2);
754
755   y1=latin_wide_high_v;
756   y2=latin_wide_low_v;
757
758   push_stroke((z1-z2)
759     shifted -centre_pt scaled (tsu_punct_size/100) shifted centre_pt,
760     (2,2)-(2,2));
761   set_bobrush(0,brpunct);
762   expand_pbox;
763 enddef;

```

PUNC

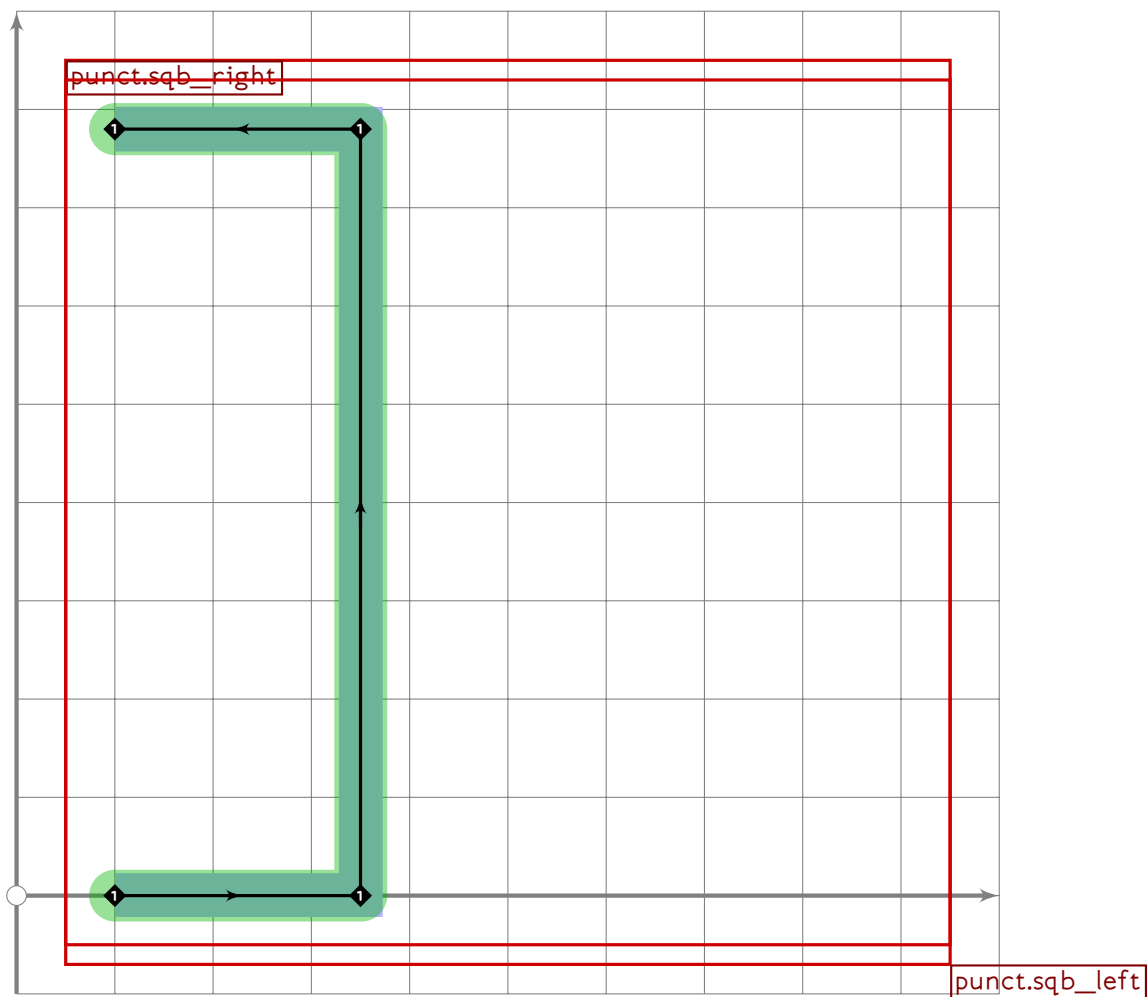


```

764
765 vardef punct.sqb_left =
766   push_pbox__toexpand("punct.sqb_left");
767   push_stroke((900,780)-
768     (900-2.5*tsu_punct_size,780)-
769     (900-2.5*tsu_punct_size,0)-
770     (900,0),
771     (2,2)-(2,2)-(2,2)-(2,2));
772   set_bobrush(0,brpunct);
773   set_bosize(0,90);
774   set_botip(0,1,1);
775   set_botip(0,2,1);
776   expand_pbox;
777 enddef;

```

U+FF3D
tsuku.uniFF3D

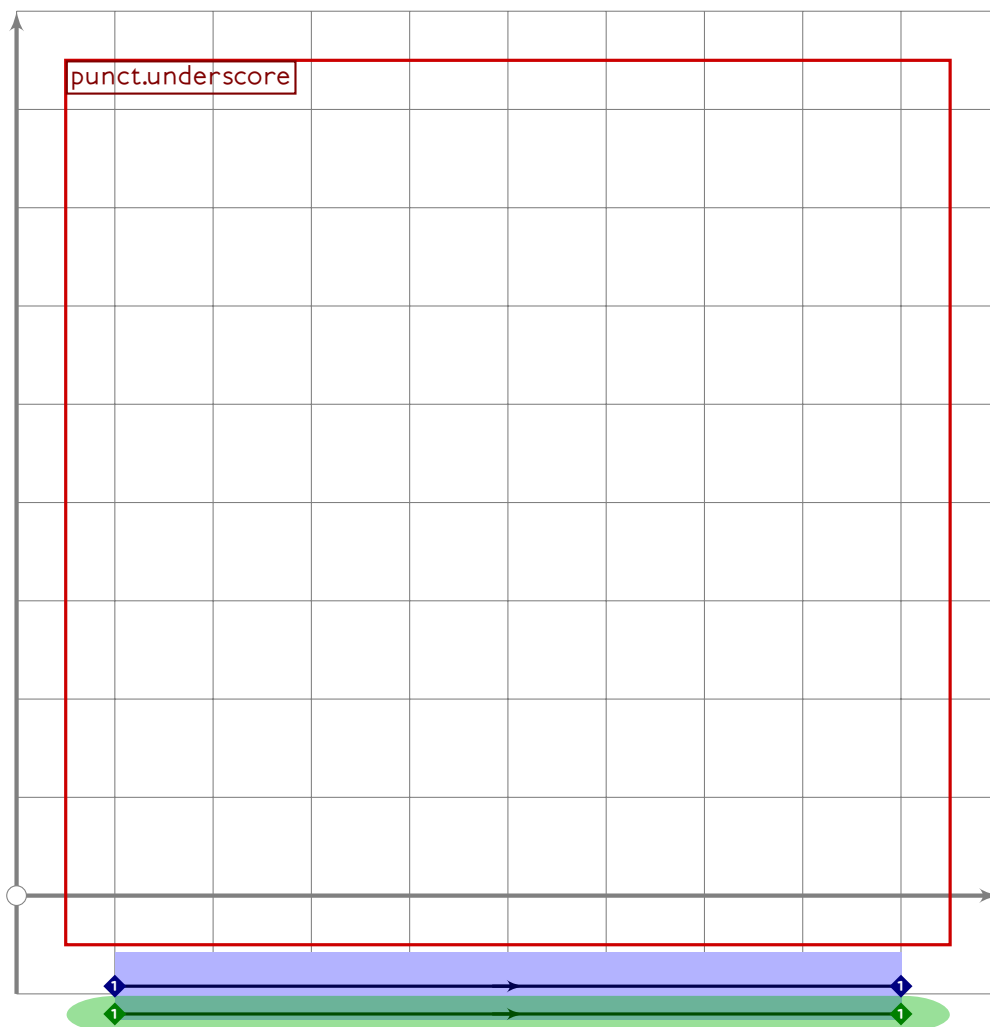


```

778
779 vardef punct.sqb_right =
780   push_pbox_toexpand("punct.sqb_right");
781   tsu_xform(identity rotatedaround (centre_pt,180))
782   (punct.sqb_left);
783   expand_pbox;
784 enddef;
785
786 vardef punct.times(expr t) =
787   push_stroke(((1,-1)-(1,1)) transformed t,(2,2)-(2,2));
788   set_bobrush(0,brpunct);
789   set_bosize(0,90);
790   push_stroke(((1,1)-(1,-1)) transformed t,(2,2)-(2,2));
791   set_bobrush(0,brpunct);
792   set_bosize(0,90);
793
794   push_pbox_explicit("punct.times",
795     identity shifted (-0.5,-0.5) scaled 24 transformed t);
796 enddef;

```

PUNC

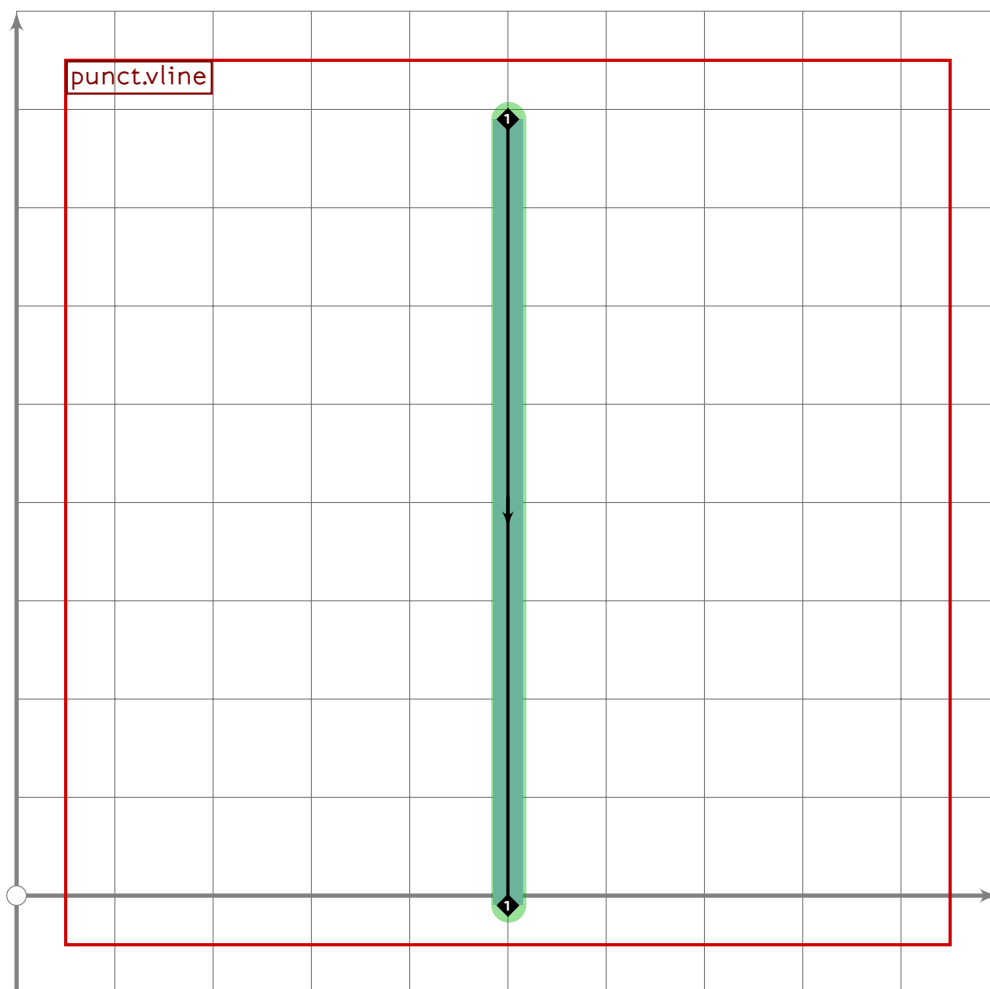


```

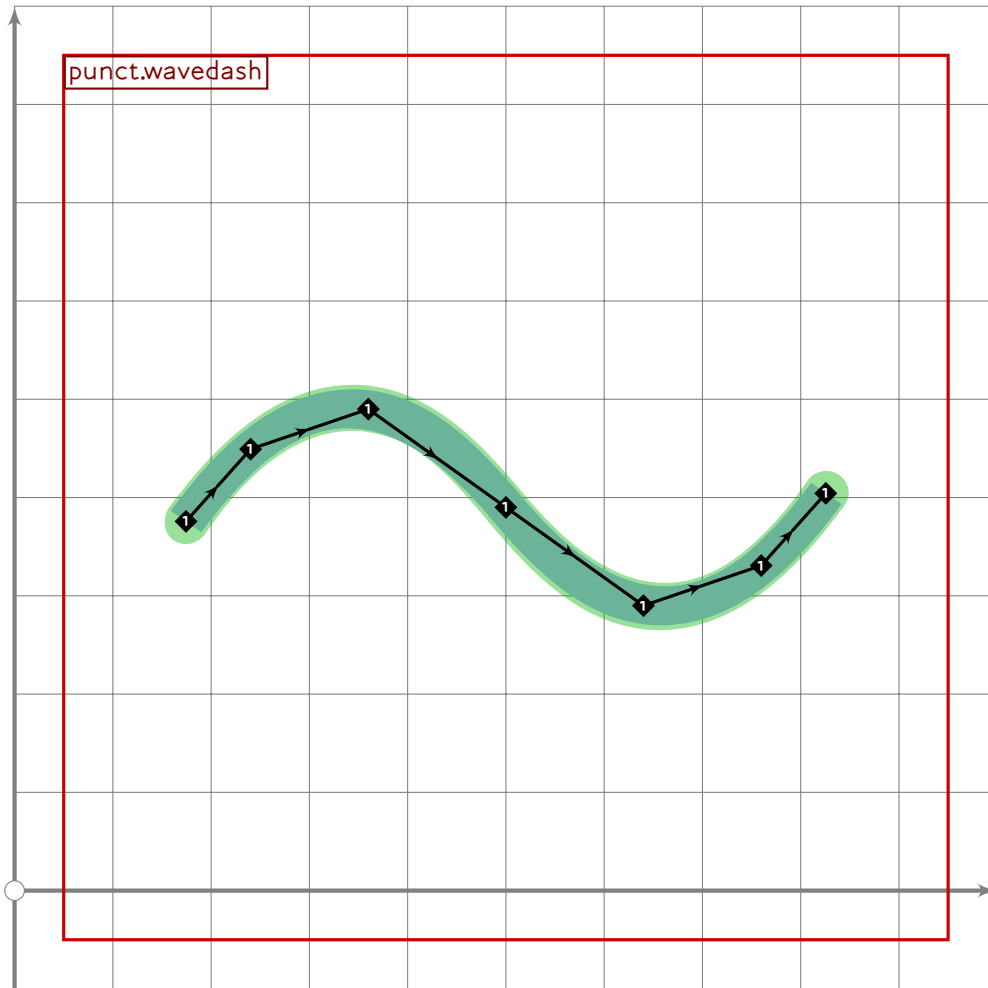
797
798 vardef punct.underscore =
799   push_pbox__toexpand("punct.underscore");
800   push_stroke((100,0.3[latin_wide_desc_h,latin_wide_low_h])–
801     (900,0.3[latin_wide_desc_h,latin_wide_low_h]),
802     (2,2)–(2,2));
803 % set_bobrush(0,brpunct);
804 set_bosize(0,90);
805 expand_pbox;
806 enddef;

```

U+FF5C
tsuku.uniFF5C



```
807
808 vardef punct.vline =
809   push_pbox__toexpand("punct.vline");
810   push_stroke((500,690+tsu_punct_size)-(500,90-tsu_punct_size),
811     (1.6,1.6)-(1.6,1.6));
812   set_bobrush(0,brpunct);
813   set_bosize(0,90);
814   expand_pbox;
815 enddef;
```

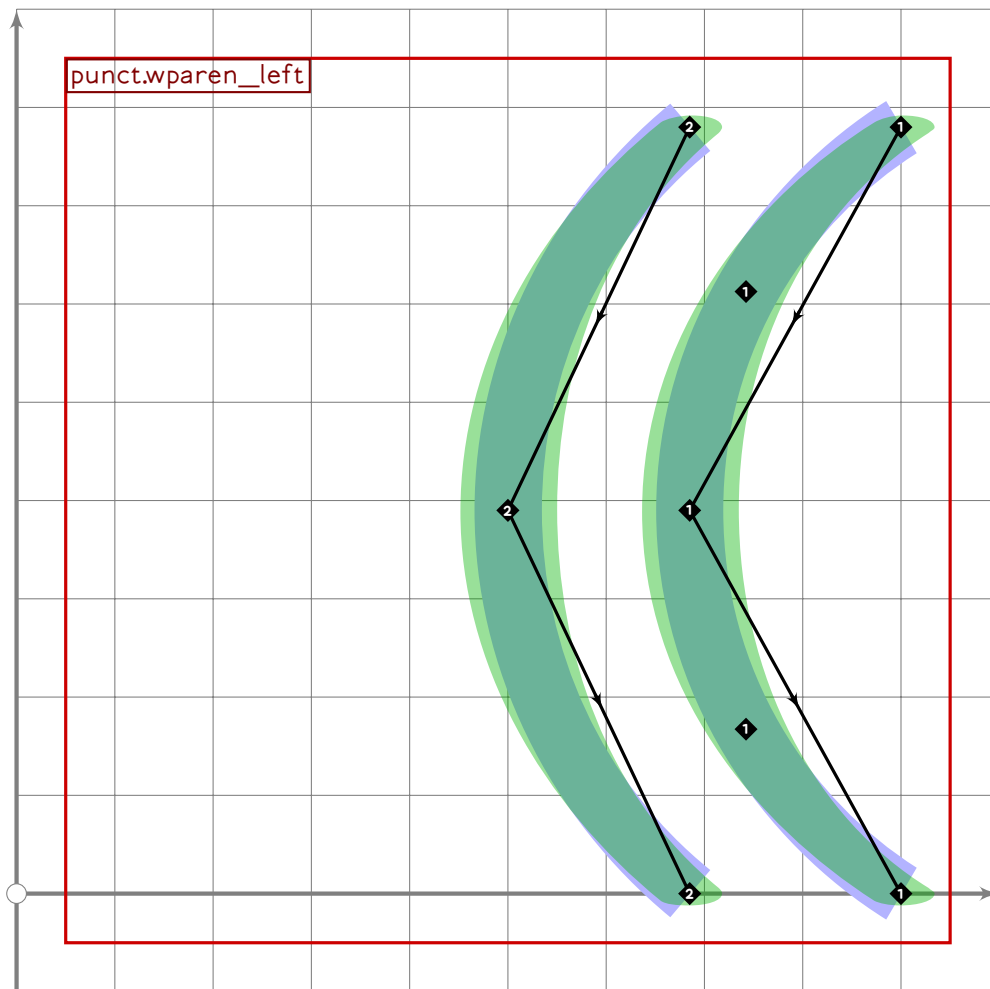



```

816
817 vardef punct.wavedash =
818   push_pbox__toexpand("punct.wavedash");
819   push_stroke((({-3.5,-0.5}{curl 0}{-14,1}..(0,0)..(14,-1)..
820     {curl 0}{3.5,0.5})) scaled tsu_punct_size shifted centre_pt,
821     (0.7,2.7)-(1.7,1.7)-(1.7,1.7)-(1.7,1.7)-(1.7,1.7)-
822     (1.7,1.7)-(0.7,2.7));
823   replace_strokep(0)(insert_nodes(oldp)(0.5,3.5));
824   set_bobrush(0,brpunct);
825   expand_pbox;
826 enddef;

```

U+FF5F
tsuku.uniFF5F

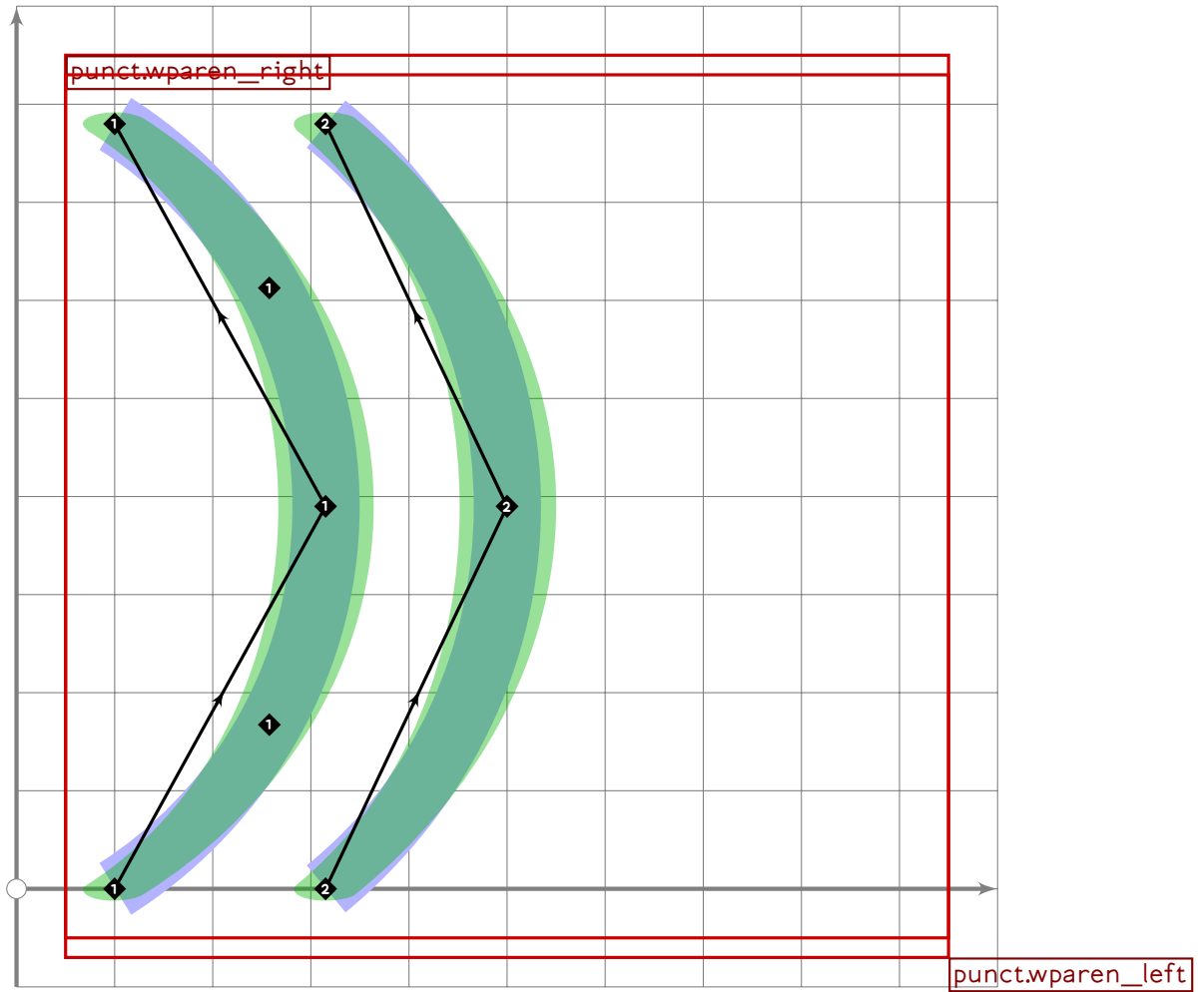


```

827
828 vardef punct.wparen_left =
829   push_pbox__toexpand("punct.wparen_left");
830
831   push_stroke((900,780)..(900-2.15*tsu_punct_size,390)..(900,0),
832     (1.5,1.5)-(2,2)-(1.5,1.5));
833   set_bosize(0,90);
834
835   push_stroke((900-2.15*tsu_punct_size,780)..
836     (900-4*tsu_punct_size,390)..
837     (900-2.15*tsu_punct_size,0),
838     (1.5,1.5)-(2,2)-(1.5,1.5));
839   set_bosize(0,90);
840   expand_pbox;
841 enddef;

```

PUNC



```

842
843 vardef punct.wparen_right =
844   push_pbox_toexpand("punct.wparen_right");
845   tsu_xform(identity rotatedaround (centre_pt,180))
846   (punct.wparen_left);
847   expand_pbox;
848 enddef;

```

serif.mp

```

1 %
2 % Serifs for Tsukurimashou
3 % Copyright (C) 2011, 2012, 2013, 2021 Matthew Skala
4 %
5-29 [Standard copyright notice]
30
31 inclusion_lock(serif);
32
33 

---


34
35 vardef tsu_serif.latin.lrcore(expr bst,plp,dlp,l,bts,bos,b) =
36   serif:=serif scaled 0.5 yscaled tsu_brush_shape[b];
37   serif:=serif xscaled ((1+3*xxpart tsu_rescaling_xf)/4)
38     yscaled bos scaled bts;
39   glstk[ngls]:=regenerate(serif shifted plp);
40   ngls:=ngls+1;
41 enddef;
42
43 vardef tsu_serif.latin.left(expr bst,plp,dlp,l,bts,bos,b) =
44   begingroup
45     save serif;
46     path serif;
47     if sharp_corners:
48       serif:=(-serif_size,1)-(-serif_size,1)-
49         (0,-1)-(0,1)-cycle;
50     else:
51       serif:=(-serif_size,1){left}..{right}(-serif_size,1)-
52         (0,-1)-(0,1)-cycle;
53     fi;
54     tsu_serif.latin.lrcore(bst,plp,dlp,l,bts,bos,b);
55   endgroup;
56 enddef;
57
58 vardef tsu_serif.latin.right(expr bst,plp,dlp,l,bts,bos,b) =
59   begingroup
60     save serif;
61     path serif;
62     if sharp_corners:
63       serif:=(0,1)-(0,-1)-
64         (serif_size,1)-(serif_size,1)-cycle;
65     else:
66       serif:=(0,1)-(0,-1)-
67         (serif_size,1){right}..{left}(serif_size,1)-cycle;
68     fi;
69     tsu_serif.latin.lrcore(bst,plp,dlp,l,bts,bos,b);
70   endgroup;

```

```

71 enddef;
72
73 vardef tsu_serif.latin.leftright(expr bst,plp,dlp,l,bts,bos,b) =
74   begingroup
75     save serif,xoffs;
76     path serif;
77     if sharp_corners:
78       serif:=(-serif_size,1)-(-serif_size,1)-
79         (serif_size,1)-(serif_size,1)-cycle;
80     else:
81       serif:=(-serif_size,1){left}{right}(-serif_size,1)-
82         (serif_size,1){right}{left}(serif_size,1)-cycle; fi;
83     numeric xoffs;
84     xoffs=xpart (dlp/abs(dlp));
85     tsu_serif.latin.lrcore(bst,
86       plp+if l=0: right else: left fi*50*sbrush_long*xoffs,
87       dlp,l,bts,bos,b);
88   endgroup;
89 enddef;
90
91
92
93 vardef tsu_serif.mincho.corner(expr bst,plp,dlp,l,bts,bos,b) =
94   begingroup
95     save serif;
96     path serif;
97     serif:=(-1,-0.3)..(0.25,-1.3)..(1,-1.2)..tension 1.2..
98       (1,0.6)..(-0.25,1.3)..(-1,1.2)..tension 1.2..cycle;
99     serif:=serif yscaled sqrt(tsu_brush_shape[b])
100       rotated tsu_brush_angle[b]
101       scaled (bts*0.43*mincho_blob_size);
102     glstk[nxls]:=regenerate(serif shifted plp);
103     nxls:=nxls+1;
104   endgroup;
105 enddef;
106
107 vardef tsu_serif.mincho.ulpoint(expr bst,plp,dlp,l,bts,bos,b) =
108   begingroup
109     save serif;
110     path serif;
111     serif:=(-1.5,2.7)..tension 2..(-1,0)..(-0.4,-0.3)..tension 1.5..
112       (0.707,0)..(0.2,0.6)..(-0.1,1.1)..tension 2..(-1.2,2.9)..cycle;
113     serif:=serif yscaled sqrt(tsu_brush_shape[b])
114       rotated tsu_brush_angle[b]
115       scaled (bts*0.5*mincho_blob_size);
116     glstk[nxls]:=regenerate(serif shifted
117       (plp+(-1,0)*0.25*bts*(xpart dlp/abs(dlp))));
118     nxls:=nxls+1;

```

```

119 endgroup;
120 enddef;
121
122 vardef tsu_serif.mincho.triangle(expr bst,plp,dlp,l,bts,bos,b) =
123   begingroup
124     save serif;
125     path serif;
126     serif=(-1.2,0)..(0,-0.8)..(1.2,0.4)..tension 2..(0.2,1.3)..
127       (-0.2,1.3)..tension 2..cycle;
128     serif:=serif yscaled sqrt(tsu_brush_shape[b])
129       rotated tsu_brush_angle[b]
130       scaled (bts*0.5*mincho_blob_size);
131     glstk[nxls]:=regenerate(serif shifted (plp+0.25*bts*dlp/abs(dlp)));
132     nxls:=nxls+1;
133   endgroup;
134 enddef;
135
136 vardef tsu_serif.mincho.lpoint(expr bst,plp,dlp,l,bts,bos,b) =
137   begingroup
138     save serif;
139     path serif;
140     serif=(-2.1,-1.9)..(-1.8,2.1)..tension 2..(-0.1,-1.2)..(0.2,-1.1)..
141       (0.707,0.707)..(-1,-0.3)..tension 2..cycle;
142     serif:=serif yscaled sqrt(tsu_brush_shape[b])
143       rotated tsu_brush_angle[b]
144       scaled (bts*0.5*mincho_blob_size);
145     glstk[nxls]:=regenerate(serif shifted plp);
146     nxls:=nxls+1;
147   endgroup;
148 enddef;
149
150 vardef tsu_serif.mincho.lpoint(expr bst,plp,dlp,l,bts,bos,b) =
151   begingroup
152     save serif;
153     path serif;
154     serif=(-1.5,1.7)..tension 2..(-1,0)..(-0.4,0.3)..tension 1.8..
155       (0.707,0.2)..(0.2,0.8)..(-0.4,1.1)..tension 2..(-1.2,2.0)..cycle;
156     serif:=reverse serif reflectedabout ((-1,1),(1,-1))
157       xyscaled (0.7,0.9) shifted (0.3,-0.3);
158     serif:=serif yscaled sqrt(tsu_brush_shape[b])
159       rotated tsu_brush_angle[b]
160       scaled (bts*mincho_blob_size);
161     glstk[nxls]:=regenerate(serif shifted plp);
162     nxls:=nxls+1;
163   endgroup;
164 enddef;
165
166 vardef tsu_serif.mincho.ktriangle(expr bst,plp,dlp,l,bts,bos,b) =

```

```

167 begingroup;
168   save serif,x,y,t,q;
169   path serif;
170   transform t;
171   (0,0) transformed t=(0,0);
172   right transformed t=(dlp/abs(dlp));
173   z1=(dlp/abs(dlp)) rotated 90;
174   if y1<-x1: x1:=-x1; y1:=-y1; fi;
175   up transformed t=z1;
176   q1:=2;
177   q2:=q1+0.5;
178   q3:=0.5*q2/q1;
179   q5:=0.5+q3;
180   z2=(0,q1);
181   z3=(-q3,q5);
182   z4=(q3,q5);
183   serif:=(0.1[z2,z4]){z2-z4}{z3-z2}(0.1[z2,z3])--
184     (0.1[z3,z2]){z3-z2}{z4-z3}(0.1[z3,z4])--
185     (0.1[z4,z3]){z4-z3}{z2-z4}(0.1[z4,z2])--cycle
186     shifted (0,-0.19)
187     transformed t yscaled tsu_brush_shape[b]
188     rotated tsu_brush_angle[b]
189     scaled (bts*bos*0.94*mincho_blob_size) shifted plp;
190   if (xxpart t)*(yyvspart t)<0: serif:=reverse serif; fi;
191   glstk[ngls]:=regenerate(serif);
192   ngls:=ngls+1;
193 endgroup;
194 endif;
195
196 vardef tsu_serif.mincho.khellipse(expr bst,plp,dlp,l,bts,bos,b) =
197   begingroup;
198     save serif,x,y,t,q;
199     path serif;
200     transform t;
201     (0,0) transformed t=(0,0);
202     right transformed t=(dlp/abs(dlp));
203     z1=(dlp/abs(dlp)) rotated 90;
204     up transformed t=z1;
205     t:=t rotated -tsu_brush_angle[b] yscaled tsu_brush_shape[b]
206       rotated tsu_brush_angle[b] scaled (bts*bos*0.99*mincho_blob_size)
207       shifted plp;
208     z2=(0.5[x3,x4],1);
209     if l=0:
210       z3=(-0.5,0);
211       z4=(1.3,0);
212     else:
213       z3=(-1.3,0);
214       z4=(0.5,0);

```

```

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

```



```

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

```