

作りましょう 0.10

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 0.10

Parametric Font Family
Proofs and pretty-printed
source code

Matthew Skala

mskala@ansuz.sooke.bc.ca

2017年11月17日

November 17, 2017

Proofs and pretty-printed source code for Tsukurimashou
Copyright © 2011, 2012, 2013, 2014, 2015 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	124
	dakuten.mp	137
	enclosed.mp	138
	genjimon.mp	140
	hiragana.mp	152
	Hiragana Vowels	152
	Hiragana Kakikukeko/Gagigugego	157

Hiragana Sashisuseso/Zajizuzezo	162
Hiragana Tachitsuteto/Dajizudedo	168
Hiragana Naninuneno	173
Hiragana Hahifuheho/Babibubebo/Papipupepo	179
Hiragana Mamimumemo	184
Hiragana Yayuyo	190
Hiragana Rarirurero	192
Hiragana Wawiwewo/N/Iteration	198
iching.mp	206
katakana.mp	208
Katakana Vowels	208
Katakana Kakikukeko/Gagigugego	212
Katakana Sashisuseso/Zajizuzezo	217
Katakana Tachitsuteto/Dajizudedo	222
Katakana Naninuneno	227
Katakana Hahifuheho/Babibubebo/Papipupepo	232
Katakana Mamimumemo	238
Katakana Yayuyo	242
Katakana Rarirurero	245
Katakana Wawiwewo/N/Iteration	251
latin.mp	257
numerals.mp	346
ogonek.mp	357
punct.mp	369
serif.mp	411
 II Shared kyouiku kanji	 417
gradeone.mp	419
gradetwo.mp	490
gradethree.mp	592
gradefour.mp	670
gradefive.mp	746
gradesix.mp	786
 III Other shared kanji	 805
bottomrad.mp	807
leftrad.mp	811

radical.mp	821
rightrad.mp	866
toprad.mp	871
gradeeight.mp	878
gradenine.mp	943
gradeten.mp	985
rare.mp	986

IV U+0000 to U+0FFF 1059

tsuku-00.mp	1061
Ascii	1061
Latin-1 Extra Characters	1155
Accented Latin	1176
tsuku-01.mp	1234
Latin Extended A Uppercase	1234
Latin Extended A Lowercase	1289
Latin Extended A Other	1347
tsuku-02.mp	1349
Latin Extended B	1349
Spacing Modifier Letters	1350
tsuku-03.mp	1359
Combining Diacritical Marks	1359

V U+1000 to U+2FFF 1375

tsuku-20.mp	1377
General Punctuation	1377
tsuku-21.mp	1394
Symbols Required By Mes-1	1394
tsuku-24.mp	1400
Circled Numerals	1400
Circled Latin And Zero	1420
Inverted Circled Numerals	1473
Doubly Circled Numerals	1483
One More Inverted Circled Numeral	1493
tsuku-25.mp	1495
Geometric Shapes	1495
tsuku-26.mp	1497

I Ching	1497
tsuku-27.mp	1514
Inverted Circled Numerals	1514
tsuku-2e.mp	1525
Cjk Radicals Supplement	1525
tsuku-2f.mp	1563
Kangxi Radicals	1563
Ideographic Description Characters	1683
 VI U+3000 to U+4DFF	 1697
tsuku-30.mp	1699
Ideographic Symbols And Punctuation	1699
Hiragana	1719
Katakana	1764
tsuku-31.mp	1811
Phonetic Extensions For Ainu	1811
tsuku-32.mp	1828
Circled Numerals	1828
Circled Katakana	1858
tsuku-34.mp	1906
tsuku-35.mp	1907
tsuku-4d.mp	1908
I Ching	1908
 VII U+4E00 to U+5BFF	 1973
tsuku-4e.mp	1975
tsuku-4f.mp	2007
tsuku-50.mp	2064
tsuku-51.mp	2079
tsuku-52.mp	2091
tsuku-53.mp	2125
tsuku-54.mp	2146
tsuku-55.mp	2154
tsuku-56.mp	2159
tsuku-57.mp	2167
tsuku-58.mp	2179
tsuku-59.mp	2190

tsuku-5a.mp	2205
tsuku-5b.mp	2210

VIII U+5C00 to U+65FF	2233
tsuku-5c.mp	2235
tsuku-5d.mp	2254
tsuku-5e.mp	2262
tsuku-5f.mp	2297
tsuku-60.mp	2366
tsuku-61.mp	2494
tsuku-62.mp	2547
tsuku-63.mp	2565
tsuku-64.mp	2578
tsuku-65.mp	2581

IX U+6600 to U+6AFF	2609
tsuku-66.mp	2611
tsuku-67.mp	2622
tsuku-68.mp	2746
tsuku-69.mp	2859
tsuku-6a.mp	2948

X U+6B00 to U+83FF	3003
tsuku-6b.mp	3005
tsuku-6c.mp	3017
tsuku-6d.mp	3066
tsuku-6e.mp	3082
tsuku-6f.mp	3098
tsuku-70.mp	3105
tsuku-71.mp	3111
tsuku-72.mp	3118
tsuku-73.mp	3135
tsuku-74.mp	3138
tsuku-75.mp	3142
tsuku-76.mp	3154
tsuku-77.mp	3161

tsuku-78.mp	3170
tsuku-79.mp	3175
tsuku-7a.mp	3228
tsuku-7b.mp	3276
tsuku-7c.mp	3291
tsuku-7d.mp	3296
tsuku-7e.mp	3321
tsuku-7f.mp	3325
tsuku-80.mp	3332
tsuku-81.mp	3342
tsuku-82.mp	3347
tsuku-83.mp	3379

XI U+8400 to U+9FFF	3393
tsuku-84.mp	3395
tsuku-85.mp	3399
tsuku-86.mp	3402
tsuku-87.mp	3403
tsuku-88.mp	3405
tsuku-89.mp	3435
tsuku-8a.mp	3449
tsuku-8b.mp	3540
tsuku-8c.mp	3575
tsuku-8d.mp	3597
tsuku-8e.mp	3601
tsuku-8f.mp	3605
tsuku-90.mp	3622
tsuku-91.mp	3649
tsuku-92.mp	3661
tsuku-93.mp	3667
tsuku-95.mp	3671
tsuku-96.mp	3688
tsuku-97.mp	3714
tsuku-98.mp	3718
tsuku-99.mp	3782
tsuku-9a.mp	3786
tsuku-9b.mp	3791
tsuku-9c.mp	3794

tsuku-9e.mp	3796
tsuku-9f.mp	3799
 XII U+A000 to U+10FFFF	 3801
tsuku-f7.mp	3803
Latin Small Caps	3803
tsuku-f9.mp	3830
tsuku-ff.mp	3832
Full-Width Forms	3832
Half-Width Punctuation	3858
Half-Width Katakana	3861
tsuku-1f1.mp	3925
Squared Latin	3925
Inverse Circled Latin	3951
Inverse Squared Latin	3977
tsuku-200.mp	4004
tsuku-20a.mp	4005
tsuku-21c.mp	4006
tsuku-295.mp	4007
tsuku-f17.mp	4008
Combining Dots For I Ching	4008
Miscellaneous	4017
Tomoe Ornaments	4023
Heavy Metal Umlaut	4031
Genjimon	4048
tsuku-ff0.mp	4102
Fraction Numerators	4102
tsuku-ff1.mp	4109
Fraction Denominators	4109
 XIII Mandeubsida core	 4111
hangul.mp	4113
Jamo Combining Operations	4115
jamo-basic.mp	4123
Filler Jamo	4123
Sios/Cieuc/Chieuch Family	4123
Kiyek	4126

Nieun	4127
Tikeut	4129
Rieul	4131
Mieum	4132
Pieup	4133
Sios	4134
Ieung	4134
Cieuc	4136
Chieuch	4136
Khieukh	4137
Thieuth	4138
Phieuph	4139
Hieuh	4141
Mixed Tails	4142
Vowels	4143
jamo-extra.mp	4152
Pansios	4155
Yesieung	4155
Yeorinhieuh	4156
Chitueum And Ceongchieum Variants	4158
Kapyeoun Variants	4159
hglxtb.mp	4160
Hangul Extension B	4160
Hangul Jungseong (Vowel) Jamo Extension B	4160
Hangul Jongseong (Tail) Jamo Extension B	4183
mande-bt.mp	4233
mande-do.mp	4234
mande-sm.mp	4235
hglpage.mp	4236
mande-11.mp	4238
Hangul Choseong (Lead) Jamo	4238
Hangul Jungseong (Vowel) Jamo	4333
Hangul Jongseong (Tail) Jamo	4404
mande-31.mp	4493
Hangul Compatibility Jamo	4493
mande-a9.mp	4587
Hangul Choseong (Lead) Jamo Extended A	4587
mande-ac.mp	4617

XIV	Mandeubsida alternates	4671
	mande-ff2.mp	4673
	Hangul Jungseong (Vowel) Jamo	4673
	Hangul Jungseong (Vowel) Jamo Extension B	4744
	mande-ff3.mp	4768
	Hangul Choseong (Lead) Jamo	4768
	mande-ff4.mp	4893
	Hangul Choseong (Lead) Jamo	4893
	mande-ff5.mp	5018
	Hangul Choseong (Lead) Jamo	5018
	mande-ff6.mp	5143
	Hangul Choseong (Lead) Jamo	5143
	mande-ff7.mp	5268
	Hangul Choseong (Lead) Jamo	5268
	Hangul Choseong (Lead) Jamo Extended A	5363
XV	TsuIta	5393
	tsuita-common.mp	5395
	tsuita-at.mp	5406
	tsuita-so.mp	5407
	Additional Proofs	5409
XVI	Blackletter Lolita	5463
	bll.mp	5465
	bll-co.mp	5470
	pentacross.mp	5471
	Utilities For Pentagrams And Crosses	5471
	bll-f5c.mp	5473
	Pentagrams	5473
XVII	Kazoemashou	5493
	kazoe-se.mp	5495
	kazoe-ld4.mp	5498
	Math Bold	5498
	Math Bold Italic	5505
	Math Italic	5512

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 endif;
91 def bralternate = 1 endif;
92 def brpunct = 2 endif;
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;
```

BQ

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 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 endif;
99
100 def tsu_brush_tip_size(expr l,q,b) =
101     begingroup
102         numeric y,yy;
103         y:=ypart (point l of q);
104         if y<tsu_brush_min[b]:
105             yy:=tsu_brush_min[b];
106         elseif y>tsu_brush_max[b]:
107             yy:=tsu_brush_max[b];
108         else:
109             yy:=y;
110         fi;
111         yy
112     endgroup
113 endif;
114
115 def tsu_brush_tip(expr l,p,q,bsi,is_start,is_end,b) =
116     begingroup
117         numeric y;
118         y=tsu_brush_tip_size(l,q,b);

```

```

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

```

```

167 rescale_to.right:=500-narrow_margin;
168 rescale_to.top:=latin_vcentre;
169 rescale_to.bottom:=0;
170
171 rescale_skew:=0;
172
173 width_curve:=((-1,-1)-(100,100)).(780,410)..{\right}(2000,1000);
174 \enddef;
175
176 \def tsu_rescale_half_katakana =
177   rescale_from.left:=wide_margin;
178   rescale_from.right:=1000-wide_margin;
179   rescale_from.top:=700;
180   rescale_from.bottom:=0;
181
182   rescale_to.left:=narrow_margin;
183   rescale_to.right:=500-narrow_margin;
184   rescale_to.top:=670;
185   rescale_to.bottom:=30;
186
187   rescale_skew:=8;
188
189   width_curve:=((-1,-1)-(100,100)).(860,440)..{\right}(2000,1000);
190 \enddef;
191
192 \def tsu_rescale_double =
193   rescale_from.left:=narrow_margin;
194   rescale_from.right:=500-narrow_margin;
195   rescale_from.top:=latin_vcentre;
196   rescale_from.bottom:=0;
197
198   rescale_to.left:=wide_margin;
199   rescale_to.right:=1000-wide_margin;
200   rescale_to.top:=ypart centre_pt;
201   rescale_to.bottom:=latin_wide_baseline;
202
203   rescale_skew:=0;
204
205   width_curve:=(-1,-1)-(2000,2000);
206 \enddef;
207
208 \def tsu_rescale_decenter =
209   rescale_from.left:=300;
210   rescale_from.right:=700;
211   rescale_from.top:=ypart centre_pt;
212   rescale_from.bottom:=latin_wide_baseline;
213
214   rescale_to.left:=50;

```



```

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

```

```

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

```

```

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

```

```

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

```

```

407     else:
408         (whatever,whatever)
409     fi
410 fi
411 enddef;
412
413 % find the t-values of any inflection points of subpath (0,1) of p
414 vardef segment_inflections(expr p) =
415     save x,y,c;
416     numeric x[],y[],c[];
417
418     % normalize to prevent numerical misbehaviour
419     z10=(point 0 of p)+z22;
420     z11=(postcontrol 0 of p)+z22;
421     z12=(precontrol 1 of p)+z22;
422     z13=(point 1 of p)+z22;
423     z10+z11+z12+z13=(0,0);
424     c10=(abs(z10)+abs(z11)+abs(z12)+abs(z13))/4;
425     if c10<0.1:
426         c10:=0.1;
427     fi;
428     z0=z10/c10;
429     z1=z11/c10;
430     z2=z12/c10;
431     z3=z13/c10;
432
433     % abort if points are close enough to collinear
434     if (abs(x0*y1-x1*y0)<0.01) and (abs(x2*y3-x3*y2)<0.01):
435         (whatever,whatever)
436     else:
437
438         % find t-values at which  $|z'(t) \text{ cross } z''(t)|=0$ 
439         c2=y0*( -1*x1 +2*x2 -x3)
440             +y1*( x0 -3*x2 +2*x3)
441             +y2*(-2*x0 +3*x1 -x3)
442             +y3*( x0 -2*x1 +x2 );
443
444         c1=y0*( 2*x1 -3*x2 +x3)
445             +y1*(-2*x0 +3*x2 -x3)
446             +y2*( 3*x0 -3*x1 )
447             +y3*( -x0 +x1 );
448
449         c0=y0*( -x1 +x2)
450             +y1*( x0 -x2)
451             +y2*( -x0 +x1 );
452
453         z20=solve_quadratic(c2,c1,c0);
454

```

```

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

```

```

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

```

```

551   if obstackn[i]=hsmain_render:
552       scantokens obstacks[i];
553   fi;
554 fi; endfor;
555 begingroup
556   numeric i,j,k,l,st,si,ngls;
557   path bqi,p,q,glstk[];
558   ngls:=0;
559   for i=1 upto sp-1: if obstacktype[i]=otstroke:
560       if unknown obstackna.bobrush[i]:
561           obstackna.bobrush[i]:=brletter;
562       fi;
563       if (obstackna.bobrush[i]=bralternate) and not do_alteration:
564           obstackna.bobrush[i]:=brletter;
565       fi;
566 % message "suffix " & str i;
567       bqi:=obstackq[i] transformed tsu_brush_xf[obstackna.bobrush[i]];
568       s:=0;
569       for j=0 step 1 until (length obstackp[i])-1:
570           k:=j+1;
571 % message " j=" & decimal j & " thr " & decimal (xpart point j of bqi)
572 % & "/" & decimal (xpart point k of bqi);
573           if ((xpart point j of bqi)<1)
574               and ((xpart point k of bqi)>=1):
575 % message " START";
576               if (xpart point k of bqi)=1:
577                   s:=k;
578               else:
579                   s:=j+(xpart ((subpath (j,k) of bqi)
580                       intersectiontimes ((1,-infinity)
581                           -(1,infinity))));
582               fi;
583           fi;
584           if (((xpart point j of bqi)>=1) and ((xpart point k of bqi)<1))
585               or (k=length obstackp[i])):
586 % message " END";
587               if (xpart point k of bqi)>=1:
588                   t:=k;
589               else:
590                   t:=j+(xpart ((subpath (j,k) of bqi)
591                       intersectiontimes ((1,-infinity)
592                           -(1,infinity))));
593               fi;
594               if ((t-s)>0.02) and (obstackna.bosize[i]>0):
595                   p:=subpath (s,t) of obstackp[i];
596                   q:=subpath (s,t) of bqi;
597                   numeric ltime[];
598                   ltime[0]:=s;

```



```

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

```

```

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

```

```

694 % write misc. proof file stuff
695 %
696 blobcount:=0;
697 boxcount:=0;
698 for i=1 upto sp-1:
699     if obstacktype[i]=otlcblob:
700         begingroup
701             save spt,n;
702             pair spt;
703             spt:=(0,0);
704             n:=0;
705             for j=1 upto length obstackp[i]:
706                 n:=n+1;
707                 spt:=spt+(point j of obstackp[i]);
708             endfor;
709             spt:=spt/n;
710             blobcount:=blobcount+1;
711             write ("BLOBCENTRE "&(decimal blobcount)&" "
712                 &(decimal xpart spt)&" "&(decimal ypart spt)) to "proof.prf";
713         endgroup;
714     elseif obstacktype[i]=otplibbox:
715         boxcount:=boxcount+1;
716         write ("PBOX "&
717             (decimal boxcount)&" "&
718             (decimal xpart ((0,0) transformed obstackt[i]))&" "&
719             (decimal ypart ((0,0) transformed obstackt[i]))&" "&
720             (decimal xpart ((1,0) transformed obstackt[i]))&" "&
721             (decimal ypart ((1,0) transformed obstackt[i]))&" "&
722             (decimal xpart ((1,1) transformed obstackt[i]))&" "&
723             (decimal ypart ((1,1) transformed obstackt[i]))&" "&
724             (decimal xpart ((0,1) transformed obstackt[i]))&" "&
725             (decimal ypart ((0,1) transformed obstackt[i]))&" "&
726             obstacks[i]&"") to "proof.prf";
727         if known obstackba.botoexpand[i]:
728             if obstackba.botoexpand[i]:
729                 errmessage "Unexpanded PBOX: " & obstacks[i];
730             fi;
731         fi;
732     elseif obstacktype[i]=otanchor:
733         begingroup
734             save topanchor;
735             numeric topanchor;
736             topanchor:=i;
737             for j:=sp-1 downto i+1:
738                 if obstacktype[j]=otanchor:
739                     if obstackn[j]=obstackn[i]:
740                         topanchor:=j;
741                     fi;

```

```

742         fi;
743         exitif topanchor<>i;
744     endfor;
745     if topanchor=i:
746         write ("ANCHOR "&
747             (decimal obstackn[i])&" "&
748             (decimal xpart ((-35,0) transformed obstackt[i]))&" "&
749             (decimal ypart ((-35,0) transformed obstackt[i]))&" "&
750             (decimal xpart ((35,0) transformed obstackt[i]))&" "&
751             (decimal ypart ((35,0) transformed obstackt[i]))&" "&
752             (decimal xpart ((0,-35) transformed obstackt[i]))&" "&
753             (decimal ypart ((0,-35) transformed obstackt[i]))&" "&
754             (decimal xpart ((0,35) transformed obstackt[i]))&" "&
755             (decimal ypart ((0,35) transformed obstackt[i])))
756         to "proof.prf";
757     fi;
758 endgroup;
759 fi;
760 endfor;
761 endgroup;
762 enddef;
763
764 % the usual case - just render it without fitting into a circle
765 def tsu_render =
766     tsu_render_in_circle(identity scaled -1);
767 enddef;
768
769 transform tsu_xf.smallkana;
770
771 tsu_xf.smallkana = identity shifted (-500,0) scaled 5.5/8 shifted (500,0);
772
773 def tsu_xform(expr xform)(text curves) =
774     begingroup
775         save txfsp,zc,zs;
776         txfsp:=sp;
777         curves;
778         numeric zs;
779         zs:=abs(((0,0) transformed xform)
780             -((1,0) transformed xform))
781             *abs(((0,0) transformed xform)
782             -((0,1) transformed xform));
783         size_scale:=zs**0.16667;
784         for i=txfsp upto sp-1:
785             if known obstackp[i]:
786                 if known obstackba.bokeepshape[i]:
787                     if obstackba.bokeepshape[i]:
788                         pair zc;
789                         zc:=0.5[lcorner obstackp[i],urcorner obstackp[i]];

```

```

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

```

```

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

```

```
886     sbrush_short:=sbrush_width;
887     sbrush_long:=sbrush_height;
888     fi;
889 endif;
890
891 calc_mbrush_size;
```

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

```

```

188 (ypart llcorner b < ypart llcorner a) and
189 (ypart urcorner b > ypart urcorner a)
190 else: % |numeric a and pair b|
191 (a>=xpart b) and (a<=ypart b)
192 fi
193 endif;
194
195 % The macro |&&| is to be used instead of the |&| operator if the respective
196 % ends of paths coincide only approximately; using |..| instead would add
197 % unwanted tiny B\`ezier segments. The macro is somewhat "left-handed,"
198 % i.e., it does not consider the expression that follow the macro, therefore,
199 % it can be used before the 'lcycle' command; if the argument |p| of the
200 % macro |amp_amp_| is a |pair|, it is just ignored which may be
201 % considered hardly intuitive.
202 def && = amp_amp_ whatever endif;
203 tertiarydef p amp_amp_ q =
204 if not pair p:
205 (subpath(0,length(p)-1) of p) .. controls (postcontrol length(p)-1 of p)
206 and (precontrol length(p) of p) ..
207 fi
208 endif;
209
210 vardef extrapolate expr t of b = % |t| pair, |b| B\`ezier segment
211 clearxy;
212 Casteljau(xpart(t)) = point 0 of b;
213 Casteljau(1/3[xpart(t),ypart(t)]) = point 1/3 of b;
214 Casteljau(2/3[xpart(t),ypart(t)]) = point 2/3 of b;
215 Casteljau(ypart(t)) = point 1 of b;
216 z0 .. controls z1 and z2 .. z3
217 endif;
218
219 def Casteljau(expr t) =
220 t[t[t[z0,z1], t[z1,z2] ], t[t[z1,z2], t[z2,z3] ] ]
221 endif;
222
223 vardef elongation_to_times(expr ea,eb) =
224 % negative parameter values are admissible; they are meant for |pen_stroke|
225 (if ea<0: - fi 1/(abs(ea)+1), eb/(abs(eb)+1))
226 endif;
227
228 % A numerical function 'lpoint_line_dist' takes as a parameter
229 % three |pair| expressions and returns a (signed) value of the distance
230 % of the first parameter from the line defined by the other two.
231 % It is referred to in the 'lis_line1' function.
232
233 vardef point_line_dist(expr a,b,c) =
234 clearxy; save d_; d_=sqrt(length(b-c));
235 z0=a/d_; z1=b/d_; z2=c/d_;

```

```

236 (x2-x1)*(y1-y0)-(x1-x0)*(y2-y1)
237 enddef;
238
239 % The idea of calculation of a turning angle
240 % between two vectors, employed in the definition of the function
241 % 'turn_ang', is based on the following observation:
242 vardef turn_ang(expr za,zb) =
243   if (abs(za)>=1/1000) and (abs(zb)>=1/1000): % lepl may be not enough
244     angle(unitvector(za) zscaled (unitvector(zb) reflectedabout (origin,right)))
245   else: whatever fi
246 enddef;
247
248 % A Boolean function 'is_line' checks whether a given B\ezier segment
249 % is a straight line. For large segments (fonts) it makes sense to specify
250 % a numerical parameter is_line_off>=0; it defines a maximal acceptable
251 % distance of the control points of a B\ezier arc from its secant
252 % (which corresponds to the distance between the arc and the secant
253 % circa |3/4is_line_off| for a symmetric, inflexionless arcs).
254 vardef is_line(expr B) =
255   save r_; boolean r_;
256   if known is_line_off:
257     save a_;
258     a_=length((point 1 of B)-(point 0 of B));
259     r_=(a_+arclength(B))<=(a_/infinity);
260     if r_:
261       r_=(is_line_off>=abs(point_line_dist(
262         postcontrol 0 of B, point 0 of B, point 1 of B))) and
263       (is_line_off>=abs(point_line_dist(
264         precontrol 1 of B, point 0 of B, point 1 of B)));
265     fi
266   else: % backward compatibility
267     save a_,b_,c_,d_;
268     a_=length((point 1 of B)-(point 0 of B));
269     b_=length((postcontrol 0 of B)-(point 0 of B));
270     c_=length((precontrol 1 of B)-(postcontrol 0 of B));
271     d_=length((point 1 of B)-(precontrol 1 of B));
272     r_=(a_+b_+c_+d_ <= a_/infinity);
273   fi
274   r_
275 enddef;
276
277 % Abbreviations for a few simple yet useful phrases
278 def xyscaled primary p = xscaled xpart(p) yscaled ypart(p) enddef;
279 def yxscaled primary p = yscaled xpart(p) xscaled ypart(p) enddef;
280
281 % The macro linsert_nodes inserts additional nodes at given non-integer
282 % non-repeating times |t| into a given path |p|.
283 % The code would be a bit longer without 'larclength' and 'larctime'!
```

```

284 % The macro can be useful in some cases in the context of finding
285 % the envelopes of pen-stroked paths (avoiding inflection
286 % points—see below).
287 vardef insert_nodes(expr p)(text t) =
288   save j_, p_, s_, t_; path p_; p_:=p;
289   t_:=0;
290   for i_:=t:
291     if round(i_)<>i_: % ignore integer times
292       t_[incr t_]=arclength(subpath(0,i_ mod length(p_)) of p_);
293     fi
294   endfor
295   for i_:=1 upto t_:
296     s_:=arctime t_[i_] of p_;
297     if abs(round(s_)-s_)>eps: % ignore repeating times; is lepsl OK?
298       p_:=subpath (0, s_) of p_ && (subpath (s_,length p_) of p_)
299       if cycle p_: & cycle fi;
300     fi
301   endfor;
302   p_
303 enddef;
304
305 % get rid of degeneracies
306 def regenerate(expr p) =
307   if (xpart urcorner p-xpart ulcorner p<5)
308     and (ypart ulcorner p-ypart llcorner p<5):
309     p
310   else:
311     begingroup
312     save q;
313     path q;
314     for t=1 step 1 until length p:
315       if abs((point t of p)-(point (t-1) of p))>3:
316         if unknown q:
317           q:=subpath (t-1,t) of p;
318         elseif length(q)=1:
319           q:=(point 0 of q)..
320             controls (postcontrol 0 of q) and (precontrol 1 of q)..
321             (0.5[point 1 of q,point t-1 of p])..
322             controls (postcontrol t-1 of p) and (precontrol t of p)..
323             (point t of p);
324         else:
325           q:=(subpath (0,length(q)-1) of q)..
326             controls (postcontrol length(q)-1 of q)
327             and (precontrol length(q) of q)..
328             (0.5[point length(q) of q,point t-1 of p])..
329             controls (postcontrol t-1 of p) and (precontrol t of p)..
330             (point t of p);
331       fi;

```

```

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

```

Prefix And Suffix Handling

```

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

```

```

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

```



```

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

```

FNTB

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

```

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

```

```

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

```

```

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

```

```

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

```

```

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

```

```

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

```

```

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

```

```

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

```



```

804 % is shown in the following picture:
805 % \LINE{\epsfbox{\illusname.110}}
806
807 % Ad 2. The call of the macro |cut| has the form: |cut|(angle,
808 % pen)(list_of_nodes) or |cut|(pen, angle)(list_of_nodes),
809 % where “pen” and “list_of_nodes” are defined as
810 % previously. The pen parameter can be omitted which means using a default
811 % pen (|default_nib|). The macro replaces a default pen with a special
812 % “razor” pen at specified nodes. More precisely, it is a projection of a
813 % given pen in the direction of the path |p| at a given node onto a
814 % straight line going through this node under the angle specified in the
815 % respective parameter of the macro. Uf\f\f\dots\ The angle of the straight
816 % line can be defined either absolutely (with respect to the axis |x|)
817 % or—by adding a prefix ‘|rell|’—relatively to the direction of the path
818 % at a given node. From the point of view of a user, the result of the
819 % macro |cut| is “cutting” the expanded stroke with a straight
820 % line. This operation is particularly useful at the ends of a path and
821 % corresponds to the setting |linecap:=butt| in \MP{, except that in \MP{
822 % one cannot specify angles. The result of the statement
823 % \LINE{descriptioncomments
824 % |pen_stroke(cut(45)(0)|
825 % |cut(default_nib xyscaled (1,2), rel 90)(infinity))(p)(q)|
826 % \unskip}
827 % \descriptioncomments
828 % that cuts both ends and, moreover, changes a pen
829 % at the ending node is shown in the figure below
830 % (at the beginning node, the absolute angle of 45 degrees is specified,
831 % at the ending one—the relative angle of 90 degrees):
832
833 % Ad 3. The call of the macro |tip| has the form |tip|(pen,
834 % pre_elongate, post_elongate)(list_of_nodes), where “pen”
835 % and “list_of_nodes” have the same meaning as previously.
836 % In particular, a pen can be omitted. At corner nodes
837 % specified in the list of nodes, the consecutive elements of the outline
838 % are not joined with an appropriate subpath of a pen; instead, they
839 % are elongated (extrapolated) until they intersect. This process corresponds
840 % (roughly) to the \MP{ setting |linejoin:=mitered|:
841
842 % The illustration above is the result of the following call
843 % of the macro |pen_stroke| (the macro |tip| is invoked with default
844 % settings, only the number of a node is specified):
845 % |pen_stroke(tip()(3))(p)(q); draw q;|
846 % The optional parameters |pre_elongation| and |post_elongation| define how
847 % far the consecutive edges (segments) should be elongated in order to make
848 % them intersect each other (the measure is the time). If one parameter is
849 % omitted, both will receive the same value; if both are omitted, a default
850 % value, |(0.5,0.5)| (it corresponds to elongation by circa 50\%), will be
851 % used. The precise meaning of the pre- and post-elongation is defined as

```

```

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

```

```

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

```

```

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

```

```

996 enddef;
997
998
999

```

Postscript Font Generation

```

1000 % POSTSCRIPT FONT GENERATION
1001
1002 % Note that this has been stripped down a lot from the METATYPE1
1003 % original code; most of the stuff for hinting, ligature tables,
1004 % METAFONT-style proof generation, and so on has been removed
1005 % because it's irrelevant to Tsukurimashou.
1006
1007 vardef pfi_file = jobname & ".pfi" enddef;
1008 vardef pic_file = "piclist" enddef;
1009 vardef dim_file = jobname & ".dim" enddef;
1010
1011 errorstopmode; warningcheck:=-1;
1012 ignore:=whatever; process:=0; utilize:=1; store:=2; % constants for introducing
1013 let semicolon_ = ; % stores original meaning of a semicolon
1014 newinternal tracingdimens; % if |tracingdimens>0| then |dim_file| is generated
1015
1016 def write_special = % additional info to be processed by AWK
1017   special "%GLYNFO: " &
1018 enddef;
1019 vardef mtone_glyph_pfx = "MT1: glyph " & str glyph_name & ": " enddef;
1020 def mtone_message = message mtone_glyph_pfx & enddef;
1021
1022 % Macro write_tex provides contact with the
1023 % outer world. The macro contains the information about EPSes that is
1024 % used for proofing and assembling the font; must be consistent with
1025 % the definitions contained in the files 'mpform.sty' and 'mp2pf.awk'.
1026 vardef write_tex(expr name, num) =
1027   write "\EPSNAMEandNUMBER{" & name & "}{" & decimal(num) & "}"
1028   to pic_file & ".tex"
1029 enddef;
1030
1031 % The following macros are related to the operation of slanting.
1032 % In particular, they enable to keep a fixed width of a stem
1033 % after slanting.
1034 vardef slant_ang = % should be rather called "local_slant_angle"
1035   slang \ if known glyph_slanting.glyph_name: * glyph_slanting.glyph_name fi
1036 enddef;
1037 vardef slant_val = tand(slant_ang) enddef;
1038 vardef slant_preadjust(expr slope, slang) =
1039   % |if sind(angle(slope))=0: 1 else:|
1040   % | abs(sind(angle(slope))/sind(angle(cotd(angle(slope))+tand(slang),1)))|

```

```

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

```

```

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

```

```

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

```



```

1185   errhelp "The result is likely to be weird.";
1186   errmessage mtone_glyph_pfx & "strange turning number in unFill, " &
1187   decimal(turningnumber h_);
1188   fi
1189   if glyph_usage div store = 1: % storing
1190     glyph_stored.glyph_name[incr glyph_stored.glyph_name.num]=h_;
1191   fi
1192   glyph_list[incr glyph_list.num]:=round_node_values(h_ italicized);
1193   endfor;
1194 endgroup
1195 endif;
1196
1197 def fix_hsbw (expr xr,ml,mr) =
1198   glyph_shift:=round(ml); % shift = left margin
1199   glyph_width:=round(xr+ml+mr); % declared width plus margins
1200   if glyph_usage div store = 1: % storing
1201     glyph_shift.glyph_name:=glyph_shift; glyph_width.glyph_name:=glyph_width;
1202   fi
1203 endif;
1204
1205 def fix_exact_hsbw(expr xr,ml,mr) =
1206   glyph_shift:=round(ml); % shift = left margin
1207   glyph_width:=xr+ml+mr; % declared width plus margins
1208   if glyph_usage div store = 1: % storing
1209     glyph_shift.glyph_name:=glyph_shift; glyph_width.glyph_name:=glyph_width;
1210   fi
1211 endif;
1212
1213 % Macros below set PostScript and \TeX{} units; a trick with '\#'
1214 % in {\it tfm\_units\}/} proves useful in achieving compatibility
1215 % with the Knuthian fonts (e.g., it is employed in {\it logo\}/} font).
1216 % Old versions of {\it tfm\_units\}/} and {\it ps\_units\}/} are less
1217 % accurate, but are kept because of backward compatibility reasons.
1218 vardef tfm_units(text x) =
1219   save #; if known (x#): x# else: x/(1000/designsize) fi
1220 endif;
1221 vardef old_tfm_units(text x) =
1222   save #; if known (x#): x# else: x/1000*designsize fi
1223 endif;
1224
1225 vardef ps_units(expr x) = x*(1000/designsize) endif;
1226 vardef old_ps_units(expr x) = x/designsize*1000 endif;
1227
1228 def define_ps_units(text t) =
1229   forsuffices $:=t: $:=ps_units($.); endfor
1230 endif;
1231 def define_whole_ps_units(text t) =
1232   forsuffices $:=t: $:=round(ps_units($.)); endfor

```

```

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

```

```

1281 newinternal show_stroke_size; show_stroke_size:=1.5;
1282 color show_stroke_color; show_stroke_color:=red;
1283
1284 color label_dot_color, label_text_color;
1285 label_dot_color:=.8white; label_text_color:=black;
1286 newinternal label_dot_size; label_dot_size:=3bp;
1287
1288 % Begin and end of the definitions of a character glyph:
1289 def begin_skip =
1290   let endglyph = fi;
1291   let ; = end_skip semicolon_
1292   if false:
1293   enddef;
1294 def end_skip =
1295   let ; = semicolon_ semicolon_
1296   let endglyph = endglyph_;
1297   enddef;
1298
1299 def uni_name(text name) = % name is either a suffix or a string expression
1300   if is_suffix(name):
1301     name
1302   else:
1303     scantokens (begingroup
1304       save rval;
1305       string rval;
1306       rval:="" for i=1 upto length (name):
1307         & "_" & (substring (i-1,i) of (name))
1308     endfor;
1309     rval
1310   endgroup)
1311   fi
1312 enddef;
1313
1314 def glyph_name_ext = enddef;
1315 def beginglyph(text name) =
1316   %
1317   def original_glyph_name = name enddef;
1318   def glyph_name = uni_name(name) glyph_name_ext enddef; % to use in lendglyph
1319   numeric glyph_usage; glyph_usage:=glyph_usage.glyph_name;
1320   if unknown glyph_usage: expandafter begin_skip fi
1321   string ps_name; ps_name:=original_glyph_name;
1322   if unknown ps_name:
1323     errhelp "Use macro 'introduce' or 'assign_name' prior to 'beginglyph.'";
1324     errmessage "MT1: PS name not assigned to " & str glyph_name;
1325   fi
1326   if name_used(original_glyph_name):
1327     errhelp "Proceed if you wish, I'll use the second glyph description.";
1328     errmessage "MT1: double output: name " & (str glyph_name);

```

```

1329 fi
1330 if glyph_usage mod store = 1: % utilizing
1331   mark_name_used(original_glyph_name);
1332 fi
1333 numeric glyph_code, glyph_num; glyph_code:=name_to_code(original_glyph_name);
1334 if glyph_code<0: glyph_num:=500-decr(min_glyph_code); else:
1335   glyph_num:=100+glyph_code;
1336   if glyph_code>max_glyph_code: max_glyph_code:=glyph_code; fi
1337 fi
1338 %
1339 beginfig(glyph_num)
1340 if glyph_usage mod store = 1: % utilizing
1341   write_special "NAME " & ps_name & " " & decimal(glyph_code);
1342   % mpform.sty and mp2pf.awk interface
1343 % |write_tex(glyph_name, glyph_num);|
1344   write_tex(ps_name, glyph_num);
1345 fi;
1346 glyph_list.num:=label_list.num:=0;
1347 path glyph_list[\\];
1348 picture label_list[\\]; pair label_list.dot[\\];
1349 numeric glyph_llx, glyph_lly, glyph_urx, glyph_ury;
1350 numeric bitmap_scale; pair bitmap_offset;
1351 numeric glyph_shift, glyph_width, glyph_axis;
1352 save glyph;
1353 hstem_list.num:=vstem_list.num:=hstem_list.cov:=vstem_list.cov:=0;
1354 pair hstem_list[\\], vstem_list[\\];
1355 path hstem_list_segms[\\], vstem_list_segms[\\];
1356 numeric old_hinting_scheme, new_hinting_scheme;
1357 if glyph_usage div store = 1: % storing
1358   if not path glyph_stored.glyph_name[0]: % glyph_name may contain digits
1359     scantokens("path " & genericize(str glyph_stored.glyph_name) & "[ ]");
1360     scantokens("pair " & genericize(str hstem_stored.glyph_name) & "[ ]");
1361     scantokens("path " & genericize(str hstem_stored_segms.glyph_name) & "[ ]");
1362     scantokens("pair " & genericize(str vstem_stored.glyph_name) & "[ ]");
1363     scantokens("path " & genericize(str vstem_stored_segms.glyph_name) & "[ ]");
1364   fi
1365   glyph_stored.glyph_name.num:=0;
1366   hstem_stored.glyph_name.num:=0; vstem_stored.glyph_name.num:=0;
1367 fi
1368 scantokens extra_beginglyph;
1369 enddef;
1370
1371 picture endglyph_picture;
1372 def endglyph =
1373   scantokens extra_endglyph;
1374   % usually, |currentpicture=nullpicture|, but if not (i.e., some
1375   % extra objects have been drawn), the picture must be shifted:
1376   endglyph_picture:=currentpicture shifted (glyph_shift,0);

```

```

1377 currentpicture:=nullpicture;
1378 if known glyph_axis: % actually, used only with stored chars
1379   glyph_axis.glyph_name:=glyph_axis;
1380 fi
1381 % fix char dimensions and write them to TFM and/or |dim_file|
1382 % independently of |glyph_usagel| (|dim_file|)
1383 % fix_tfm_data(glyph_urx+glyph_shift, glyph_ury);
1384 if glyph_usage mod store = 1: % utilizing
1385   write_special "HSBW * " & decimal(glyph_width);
1386   write_special "BEGINCHAR";
1387   ship_glyphs;
1388 endfig;
1389 else:
1390   endgroup; % ends figure without shipping it out
1391 fi
1392 enddef;
1393 let endglyph_=endglyph;
1394 string extra_beginglyph, extra_endglyph; extra_beginglyph=extra_endglyph="";
1395
1396 % Additional macros
1397 vardef fix_name_list text t =
1398   string name_list[]; numeric name_list.num; name_list.num:=0;
1399   save , ; let , = fix_name_list_; fix_name_list_ t
1400 enddef;
1401 def fix_name_list_ suffix name =
1402   ; % important semicolon!
1403   if str name<>"": fix_name_list_s_ name else: fix_name_list_e_ "" & fi
1404 enddef;
1405 def fix_name_list_s_ suffix s_name = fix_name_list_e_ (str s_name) enddef;
1406 def fix_name_list_e_ expr e_name = % name is expected to be of the string type
1407   name_list[incr name_list.num]=e_name
1408 enddef;
1409
1410 def introduce suffix name =
1411   if str name="": introduce_
1412   elseif (substring (0,1) of str name)<>"_": introduce__ name
1413   else: introduce___ name fi
1414 enddef;
1415 def introduce__ expr name = % name is expected to be a string expression
1416   introduce___ uni_name(name)
1417 enddef;
1418 vardef introduce___@#(expr usage, slanting)(text stencil) =
1419   if (unknown process_selected or known process_selected@#)
1420     and known usage and unknown ignore_selected@#:
1421     glyph_usage@#:=usage; % |ignore=whatever|, |process=0|, |utilize=1|, |store=2|
1422     if unknown glyph_ps_name@#: % set default:
1423       assign_name @# (substring (1,infinity) of (str @#));
1424     fi

```

```

1425 glyph_slanting@#:=slanting; % ignore |slant_ang| if |0|; use |slant_ang| otherwise
1426 % |stencil| can be either string (recommended) or suffix (with default
1427 % extension |".eps"| – obsolete), hence some trickery below
1428 save r_; string r_;
1429 for i_:=stencil: if string i_: r_:=i_; fi endfor
1430 if unknown r_:
1431   forsuffices i_:=stencil: r_:= str i_; endfor
1432   if r_<>"": r_:=r_ & ".eps"; fi
1433   fi
1434   if r_<>"":
1435     if not string glyph_stencil@#:
1436       scantokens("string " & genericize(str glyph_stencil@#));
1437     fi
1438     glyph_stencil@# = r_;
1439   fi
1440 fi
1441 enddef;
1442
1443 vardef assign_name@#(expr ps_name) =
1444   if not string glyph_ps_name @#:
1445     scantokens("string " & genericize(str glyph_ps_name@#));
1446   fi
1447   glyph_ps_name@#:=ps_name;
1448 enddef;
1449
1450 def standard_introduce(expr name) =
1451   introduce name (utilize+store)(1)();
1452 enddef;
1453
1454 vardef name_to_code(text name) =
1455   save res_, name_; string name_;
1456   name_:=name; res_=-1;
1457   for i:=0 upto 255: % 1-to-1 coding presumed
1458     if known code_to_name_[i]: if code_to_name_[i]=name_: res_:=i; fi fi
1459     exitif res_>-1;
1460   endfor
1461   res_
1462 enddef;
1463 def encode(text name)(expr glyph_code)=
1464   if (glyph_code<0) or (glyph_code>255):
1465     errhelp "The code must be within the range 0..255";
1466     errmessage "MT1: improper code " & decimal(glyph_code) &
1467       " ('encode' ignored)";
1468   elseif known code_to_name_[glyph_code]:
1469     errhelp "A given code can be assigned only to one name (obviously)";
1470     errmessage "MT1: repeated code for " & code_to_name_[glyph_code] &
1471       " (" & decimal(glyph_code) & "; 'encode' ignored)";
1472   else:

```

```

1473 code_to_name_[glyph_code]:=name;
1474 fi
1475 enddef;
1476 string code_to_name_[\\];
1477
1478 vardef name_used(text name) =
1479   save res_, name_; boolean res_; string name_;
1480   name_:=name; res_:=false;
1481   for i:=1 upto max_name_used: res_:=(name_used_[i]=name_); exitif res_; endfor
1482   res_
1483 enddef;
1484 def mark_name_used(text name)=
1485   name_used_[incr max_name_used]:=name;
1486 enddef;
1487 string name_used_[\\]; newinternal max_name_used;
1488
1489 vardef string_date =
1490   if day<10: "0" & fi decimal(day) & ":" &
1491   if month<10: "0" & fi decimal(month) & ":" &
1492   decimal(year)
1493 enddef;
1494
1495 def set_pfi (suffix kind) (expr val) =
1496   if known val:
1497     if (numeric val) or (string val) or (boolean val):
1498       if (numeric val) and (not numeric pf_info_set.kind):
1499         scantokens ("numeric " & genericize(str pf_info_set.kind));
1500       elseif (string val) and (not string pf_info_set.kind):
1501         scantokens ("string " & genericize(str pf_info_set.kind));
1502       elseif (boolean val) and (not boolean pf_info_set.kind):
1503         scantokens ("boolean " & genericize(str pf_info_set.kind));
1504       fi
1505       pf_info_set.kind:=val;
1506       write str kind & " : " &
1507       if string val: val
1508       elseif numeric val: decimal(val)
1509       elseif boolean val: if val: "true" else: "false" fi
1510       fi
1511       to pfi_file;
1512     else:
1513       errhelp "Proceed, I'll just ignore the setting.";
1514       errmessage "MT1: pf_info keys can only be of numeric, string " &
1515       "and boolean type";
1516     fi
1517   fi
1518 enddef;
1519
1520 def pf_info_version expr v = set_pfi(VERSION,v); enddef;

```

```

1521
1522 def pf_info_creationdate text t =
1523   begingroup
1524   save k_; k_:=0;
1525   for t_:=t: k_:=k_+1; set_pfi(CREATION_DATE, t_); exitif k_=1; endfor
1526   if k_=0: set_pfi(CREATION_DATE, string_date); fi
1527   endgroup
1528 enddef;
1529
1530 def pf_info_fontname text t =
1531   begingroup
1532   save k_; k_:=0;
1533   for t_:=t: k_:=k_+1;
1534     if k_=1: set_pfi(FONT_NAME, t_); fi
1535     if k_=2: set_pfi(FULL_NAME, t_); fi
1536     exitif k_=2;
1537   endfor
1538   if k_=1: set_pfi(FULL_NAME, pf_info_set.FONT_NAME); fi
1539   endgroup
1540 enddef;
1541
1542 def pf_info_author expr v = set_pfi(AUTHOR,v); enddef;
1543 % There is 'much ado about nothing', i.e., about the sign of descender:
1544 % in a PFB file in an 'ADL' comment, descender is positive, while in an AFM
1545 % in a 'Descender' comment – negative; we will distinguish between
1546 % the two, the more so as 'ADL' comment is not mentioned in
1547 % in the Adobe documentation {\it Adobe Type 1 Font Format}.
1548
1549 def pf_info_ascender expr v = ascender:=v; set_pfi(ASCENDER,v); enddef;
1550 def pf_info_descender expr v = descender:=v; set_pfi(DESCENDER,v); enddef;
1551
1552 def pf_info_adl text t =
1553   begingroup
1554   save k_; k_:=0;
1555   for t_:=t: k_:=k_+1;
1556     if (k_=1) and known t_: adl_ascender:=t_; set_pfi(ADL_ASCENDER,t_); fi
1557     if (k_=2) and known t_: adl_descender:=t_; set_pfi(ADL_DESCENDER,t_); fi
1558     if (k_=3) and known t_: adl_lineskip:=t_; set_pfi(ADL_LINESKIP,t_); fi
1559     exitif k_=3;
1560   endfor
1561   endgroup
1562 enddef;
1563
1564 def pf_info_underline text t =
1565   begingroup
1566   save k_; k_:=0;
1567   for t_:=t: k_:=k_+1;
1568     if k_=1: set_pfi(UNDERLINE_POSITION,t_); fi

```



```

1569   if k_=2: set_pfi(UNDERLINE_THICKNESS,t_); fi
1570   exitif k_=2;
1571   endfor
1572 endgroup
1573 enddef;
1574
1575 def pf_info_pfm text t =
1576 % parameters: name, bold (0 or 1), italic (0 or 1), char set;
1577 % each of them can be either known or unknown or "*" (which means unknown);
1578 % the last parameter can be either numeric or string representation of
1579 % a valid Perl numeric value (e.g., "0xFF" means 255).
1580 begingroup
1581   save k_; k_:=0;
1582   for t_:=t: k_:=k_+1;
1583     if (k_=1) and known t_: set_pfi(PFM_NAME,t_); fi
1584     if (k_=2) and known t_: set_pfi(PFM_BOLD,t_); fi
1585     if (k_=3) and known t_: set_pfi(PFM_ITALIC,t_); fi
1586     if (k_=4) and known t_: set_pfi(PFM_CHARSET,t_); fi
1587     exitif k_=4;
1588   endfor
1589 endgroup
1590 enddef;
1591
1592 def pf_info_fixedpitch expr v = set_pfi(FIXED_PITCH,v); enddef;
1593 def pf_info_capheight expr v = uc_height:=v; set_pfi(CAPHEIGHT,v); enddef;
1594 def pf_info_weight expr v = set_pfi(WEIGHT,v); enddef;
1595 def pf_info_stdvstem expr v = set_pfi(STDVW,v); enddef;
1596 def pf_info_stdhstem expr v = set_pfi(STDHW,v); enddef;
1597 def pf_info_forcebold expr v = set_pfi(FORCE_BOLD,v); enddef;
1598
1599 % TeX-related font info (fontdimens and headerbytes):
1600 def pf_info_fontdimen text t = % exceptionally, TFM units expected
1601   begingroup
1602     save i_, k_, b_; boolean b_;
1603     k_:=0;
1604     if true for t_:=t: hide(k_:=k_+1) and known t_ endfor and (k_<=3):
1605       k_:=0;
1606       for t_:=t: k_:=k_+1;
1607         if k_:=1:
1608           i_:=t_;
1609           % |b| means "we are ready to override (possibly) the previous value
1610           % of a font parameter unless we are inside |complete_tfm_info| and
1611           % then we want to set only a 'virgin' value."
1612           b_:=unknown completing_tfm_info or unknown pf_info_set.FONT_DIMEN[i_];
1613         fi
1614         if b_ and (k_=2): set_pfi(FONT_DIMEN[i_],t_); fontdimen i_: t_; fi
1615         if b_ and (k_=3): set_pfi(DIMEN_NAME[i_],t_); fi
1616       endfor

```

```

1617   if b_ and (k_=2): set_pfi(DIMEN_NAME[i_],"(unknown fontdimen name)"); fi
1618   else:
1619     errhelp "Proceed, I'll just ignore TFM fontdimen settings.";
1620     errmessage "MT1: invalid TFM fontdimen data";
1621   fi
1622 endgroup
1623 enddef;
1624 def pf_info_headerbyte text t =
1625   begingroup
1626     save i_, k_; k_:=0;
1627     if true for t_:=t: hide(k_:=k_+1) and known t_ endfor and (k_=2):
1628       k_:=0;
1629       for t_:=t: k_:=k_+1;
1630         if k_=1: i_:=t_; fi
1631         if k_=2:
1632           set_pfi(HEADER_BYTE[i_],if numeric t_: decimal(t_) else: t_ fi);
1633           if i_=9: % encoding scheme, e.g., |"TEX TEXT"|
1634             headerbyte 9: BCPL_string(t_,40); fi
1635           if i_=49: % font family, e.g., |"CMR"|
1636             headerbyte 49: BCPL_string(t_,20); fi
1637           if i_=72: % family member number, which should be between 0 and 255
1638             headerbyte 72: t_; fi
1639         fi
1640       endfor
1641     else:
1642       errhelp "Proceed, I'll just ignore TFM headerbyte settings.";
1643       errmessage "MT1: invalid TFM headerbyte data";
1644     fi
1645   endgroup
1646 enddef;
1647 def pf_info_designsize expr v = % |designsize| is special
1648   designsize:=v; set_pfi(DESIGN_SIZE,decimal(v) & " (in points)");
1649 enddef;
1650 def pf_info_italicangle expr v =
1651   begingroup
1652     save tfm_units; vardef tfm_units(text x) = c enddef;
1653     slang:=v; set_pfi(ITALIC_ANGLE,-v);
1654     pf_info_fontdimen 1, if known slant: slant else: tand(slang) fi, "(slant)";
1655   endgroup
1656 enddef;
1657 def pf_info_space text t = % three in one
1658   begingroup
1659     save k_; k_:=0;
1660     for t_:=t: k_:=k_+1;
1661       if (designsize<>0) and known t_:
1662         if k_=1:
1663           space:=t_; pf_info_fontdimen 2, tfm_units(space), "(space)";
1664         elseif k_=2:

```

```

1665     space_stretch:=t_; pf_info_fontdimen 3, tfm_units(space_stretch),
1666     "(space stretch)";
1667     elseif k_=3:
1668     space_shrink:=t_; pf_info_fontdimen 4, tfm_units(space_shrink),
1669     "(space shrink)";
1670     fi
1671     fi
1672     exitif k_=3;
1673 endfor
1674 endgroup
1675 enddef;
1676 def pf_info_normal_space text t =
1677 begingroup
1678     save k_; k_:=0;
1679     if true for t_:=t: hide(k_:=k_+1) endfor and (k_<=2):
1680     k_:=0;
1681     for t_:=t: k_:=k_+1;
1682     if (k_=1) and known t_: space:=t_; fi
1683     if (k_=2) and known t_: % |t_| is expected to be in TFM units
1684     pf_info_fontdimen 2, t_, "(space)";
1685     fi
1686     endfor
1687     if (k_=1) and (designsize<>0) and known space:
1688     pf_info_fontdimen 2, tfm_units(space), "(space)";
1689     fi
1690     fi
1691 endgroup
1692 enddef;
1693 def pf_info_space_stretch text t =
1694 begingroup
1695     save k_; k_:=0;
1696     if true for t_:=t: hide(k_:=k_+1) endfor and (k_<=2):
1697     k_:=0;
1698     for t_:=t: k_:=k_+1;
1699     if (k_=1) and known t_: space_stretch:=t_; fi
1700     if (k_=2) and known t_: % |t_| is expected to be in TFM units
1701     pf_info_fontdimen 3, t_, "(space stretch)";
1702     fi
1703     endfor
1704     if (k_=1) and (designsize<>0) and known space_stretch:
1705     pf_info_fontdimen 3, tfm_units(space_stretch), "(space stretch)";
1706     fi
1707     fi
1708 endgroup
1709 enddef;
1710 def pf_info_space_shrink text t =
1711 begingroup
1712     save k_; k_:=0;

```

```

1713 if true for t_:=t: hide(k_:=k_+1) endfor and (k_<=2):
1714   k_:=0;
1715   for t_:=t: k_:=k_+1;
1716     if (k_:=1) and known t_: space_shrink:=t_; fi
1717     if (k_:=2) and known t_: % |t_| is expected to be in TFM units
1718       pf_info_fontdimen 4, t_, "(space shrink)";
1719     fi
1720   endfor
1721   if (k_:=1) and (designsize<>0) and known space_shrink:
1722     pf_info_fontdimen 4, tfm_units(space_shrink), "(space shrink)";
1723   fi
1724 fi
1725 endgroup
1726 enddef;
1727 def pf_info_xheight text t =
1728 begingroup
1729   save k_; k_:=0;
1730   if true for t_:=t: hide(k_:=k_+1) endfor and (k_<=2):
1731     k_:=0;
1732     for t_:=t: k_:=k_+1;
1733       if (k_:=1) and known t_: lc_height:=t_; set_pfi(XHEIGHT, t_); fi
1734       if (k_:=2) and known t_: % |t_| is expected to be in TFM units
1735         pf_info_fontdimen 5, t_, "(xheight)";
1736       fi
1737     endfor
1738     if (k_:=1) and (designsize<>0) and known lc_height:
1739       pf_info_fontdimen 5, tfm_units(lc_height), "(xheight)";
1740     fi
1741   fi
1742 endgroup
1743 enddef;
1744 def pf_info_quad text t =
1745 begingroup
1746   save k_; k_:=0;
1747   if true for t_:=t: hide(k_:=k_+1) endfor and (k_<=2):
1748     k_:=0;
1749     for t_:=t: k_:=k_+1;
1750       if (k_:=1) and known t_: quad:=t_; fi
1751       if (k_:=2) and known t_: % |t_| is expected to be in TFM units
1752         pf_info_fontdimen 6, t_, "(quad)";
1753       fi
1754     endfor
1755     if (k_:=1) and (designsize<>0) and known quad:
1756       pf_info_fontdimen 6, tfm_units(quad), "(quad)";
1757     fi
1758   fi
1759 endgroup
1760 enddef;

```

```

1761 def pf_info_extra_space text t =
1762   begingroup
1763   save k_; k_:=0;
1764   if true for t_:=t: hide(k_:=k_+1) endfor and (k_<=2):
1765     k_:=0;
1766     for t_:=t: k_:=k_+1;
1767       if (k_:=1) and known t_: extra_space:=t_; fi
1768       if (k_:=2) and known t_: % |t_| is expected to be in TFM units
1769         pf_info_fontdimen 7, t_, "(extra space)";
1770       fi
1771     endfor
1772     if (k_:=1) and (designsize<>0) and known extra_space:
1773       pf_info_fontdimen 7, tfm_units(extra_space), "(extra space)";
1774     fi
1775   fi
1776 endgroup
1777 enddef;
1778 def pf_info_encoding text t =
1779   begingroup
1780   save k_; k_:=0;
1781   for t_:=t: k_:=k_+1;
1782     if (k_:=1) and known t_: if t_<>"": set_pfi(ENCODING_SCHEME, t_); fi fi
1783     if (k_:=2) and known t_: if t_<>"": pf_info_headerbyte 9, t_; fi fi
1784     if (k_:=3) and known t_: if t_<>"": set_pfi(ENCODING_NAME, t_); fi fi
1785     exitif k_:=3;
1786   endfor
1787   if (k_:=1) and known pf_info_set.ENCODING_SCHEME % upward compatibility
1788     and unknown pf_info_set.HEADER_BYTE9:
1789     pf_info_headerbyte 9, pf_info_set.ENCODING_SCHEME;
1790   fi
1791 endgroup
1792 enddef;
1793 def pf_info_familyname text t =
1794   begingroup
1795   save k_; k_:=0;
1796   for t_:=t: k_:=k_+1;
1797     if k_:=1: set_pfi(FAMILY_NAME, t_); fi
1798     if k_:=2: pf_info_headerbyte 49, t_; fi
1799     exitif k_:=2;
1800   endfor
1801   if k_:=1: pf_info_headerbyte 49, pf_info_set.FAMILY_NAME; fi
1802 endgroup
1803 enddef;
1804
1805 % bluezz forever...
1806 newinternal blue_fuzz, blue_scale, blue_shift;
1807 blue_fuzz:=0; % Adobe Type 1 Font Format, p. 41
1808 blue_scale:=0.0454545;

```

```

1809 blue_shift:=7;
1810
1811 % it is advisable to avoid typso whenever possible:
1812 def show_compose expr x = show_compose_ :=x; enddef;
1813 def show_fills expr x = show_fills_ :=x; enddef;
1814 def show_strokes expr x = show_strokes_ :=x; enddef;
1815 def show_paths expr x = show_paths_ :=x; enddef;
1816 def show_labels expr x = show_labels_ :=x; enddef;
1817 def show_boxes expr x = show_boxes_ :=x; enddef;
1818 def show_stems expr x = show_stems_ :=x; enddef;
1819 def show_stencils expr x = show_stencils_ :=x; enddef;
1820
1821 string extra_beginfont, extra_endfont; extra_beginfont=extra_endfont="";
1822
1823 def beginfont =
1824   min_glyph_code=max_glyph_code=0;
1825   complete_param_setting;
1826   scantokens extra_beginfont;
1827 enddef;
1828
1829 def complete_param_setting =
1830   if designsize=0: designsize:=10; fi
1831   if unknown space: space:=333; fi
1832   if unknown space_stretch: space_stretch:=round(1/2space); fi
1833   if unknown space_shrink: space_shrink:=round(1/3space); fi
1834   if unknown extra_space: extra_space:=round(1/3space); fi
1835   if unknown quad: quad:=1000; fi
1836   if unknown slang:
1837     if known slant: % compatibility with the Old Tradition...
1838       slang:=angle(1, slant);
1839     else: slang:=0; fi
1840   fi
1841   if unknown uc_height: uc_height:=750; fi
1842   if unknown lc_height: lc_height:=400; fi
1843   if unknown italic_shift: italic_shift:=-40; fi % used to be |-100|
1844   if unknown depth: depth:=-250; fi
1845   if unknown ascender: ascender:=uc_height; fi
1846   if unknown descender: descender:=depth; fi
1847   if unknown adl_ascender: adl_ascender:=uc_height; fi
1848   if unknown adl_descender: adl_descender:=-depth; fi
1849   if unknown adl_lineskip: adl_lineskip:=0; fi
1850   if unknown top_line: top_line:=adl_ascender+1/2adl_lineskip; fi
1851   if unknown bot_line: bot_line:=-adl_descender+1/2adl_lineskip; fi
1852   if unknown math_axis: math_axis:=250; fi
1853   if unknown math_rule: math_rule:=40; fi
1854   begingroup
1855     save rth_, pt_, subs_, desc_depth_, fig_height_, asc_height_;
1856     rth_:=math_rule; pt_:=100;

```

```

1857 % math symbol font parameters (defaults excerpted from cmsy10)
1858 subs_:=7/10;
1859 desc_depth_:=70/36pt_; fig_height_:=232/36pt_; asc_height_:=250/36pt_;
1860 if unknown num_one:
1861   num_one:=math_axis+3.51rth_+54/36pt_+subs_*desc_depth_; fi
1862 if unknown num_two: num_two:=math_axis+1.51rth_+30/36pt_; fi
1863 if unknown num_three: num_three:=math_axis+1.51rth_+48/36pt_; fi
1864 if unknown denom_one:
1865   denom_one:=3.51rth_+subs_*fig_height_+124/36pt_-math_axis; fi
1866 if unknown denom_two:
1867   denom_two:=1.51rth_+subs_*fig_height_+30/36pt_-math_axis; fi
1868 if unknown sup_one: sup_one:=8.99pt_-subs_*asc_height_; fi
1869 if unknown sup_two: sup_two:=8.49pt_-subs_*asc_height_; fi
1870 if unknown sup_three: sup_three:=104/36pt_; fi
1871 if unknown sub_one: sub_one:=54/36pt_; fi
1872 if unknown sub_two: sub_two:=-8.49pt_+2subs_*asc_height_+3.1rth_; fi
1873 if unknown sup_drop: sup_drop:=subs_*asc_height_-36/36pt_; fi
1874 if unknown sub_drop: sub_drop:=18/36pt_; fi
1875 if unknown delim_one: delim_one:=23.9pt_; fi
1876 if unknown delim_two: delim_two:=10.1pt_; fi
1877 % math extension font parameters (defaults excerpted from cmex10)
1878 if unknown big_op_spacing_one: big_op_spacing_one:=40/36pt_; fi;
1879 if unknown big_op_spacing_two: big_op_spacing_two:=60/36pt_; fi;
1880 if unknown big_op_spacing_three: big_op_spacing_three:=72/36pt_; fi;
1881 if unknown big_op_spacing_four: big_op_spacing_four:=216/36pt_; fi;
1882 if unknown big_op_spacing_five: big_op_spacing_five:=36/36pt_; fi;
1883 endgroup;
1884 enddef;
1885
1886 def endfont =
1887   scantokens extra_endfont;
1888   complete_pf_info;
1889   complete_tfm_info;
1890   scantokens "end";
1891 enddef;
1892
1893 def complete_pf_info =
1894   if unknown pf_info_set.DESIGN_SIZE: pf_info_designsize designsize; fi
1895   if unknown pf_info_set.VERSION: pf_info_version "0.000"; fi
1896   if unknown pf_info_set.AUTHOR: pf_info_author "Unknown"; fi
1897   if unknown pf_info_set.CREATION_DATE: pf_info_creationdate; fi
1898   if unknown pf_info_set.FAMILY_NAME: pf_info_familyname "Untitled"; fi
1899   if unknown pf_info_set.FONT_NAME: pf_info_fontname "Untitled"; fi
1900   if unknown pf_info_set.ASCENDER: pf_info_ascender ascender; fi
1901   if unknown pf_info_set.DESCENDER: pf_info_descender descender; fi
1902   if unknown pf_info_set.ADL_ASCENDER:
1903     pf_info_adl adl_ascender, whatever, whatever;
1904   fi

```

```

1905 if unknown pf_info_set.ADL_DESCENDER:
1906   pf_info_adl whatever, adl_descender, whatever;
1907 fi
1908 if unknown pf_info_set.ADL_LINESKIP:
1909   pf_info_adl whatever, whatever, adl_lineskip;
1910 fi
1911 if unknown pf_info_set.UNDERLINE_POSITION: pf_info_underline -200, whatever;
1912   fi
1913 if unknown pf_info_set.UNDERLINE_THICKNESS: pf_info_underline whatever, math_rule;
1914   fi
1915 if unknown pf_info_set.ITALIC_ANGLE: pf_info_italicangle slang; fi
1916 if unknown pf_info_set.FIXED_PITCH: pf_info_fixedpitch false; fi
1917 if unknown pf_info_set.CAPHEIGHT: pf_info_capheight uc_height; fi
1918 if unknown pf_info_set.XHEIGHT: pf_info_xheight lc_height; fi
1919 if unknown pf_info_set.WEIGHT: pf_info_weight "Normal"; fi
1920 if unknown pf_info_set.STDVW: fi % just ignore
1921 if unknown pf_info_set.STDHW: fi % just ignore
1922 if unknown pf_info_set.FORCE_BOLD: pf_info_forcebold false; fi
1923 if unknown pf_info_set.ENCODING_SCHEME:
1924   pf_info_encoding "FontSpecific", whatever;
1925 fi
1926 if unknown pf_info_set.HEADER_BYTE9:
1927   pf_info_encoding whatever, "UNSPECIFIED";
1928 fi
1929 if unknown pf_info_set.BLUE_VALUES: set_pfi(BLUE_VALUES, ""); fi
1930 if unknown pf_info_set.OTHER_BLUES: fi % just ignore
1931 if unknown pf_info_set.BLUE_FUZZ: set_pfi(BLUE_FUZZ, blue_fuzz); fi
1932 if unknown pf_info_set.BLUE_SCALE: set_pfi(BLUE_SCALE, blue_scale); fi
1933 if unknown pf_info_set.BLUE_SHIFT: set_pfi(BLUE_SHIFT, blue_shift); fi
1934 % for those who like smart (implicit) systems:
1935 if unknown no_implicit_spaces:
1936   if not name_used("space"):
1937     if unknown glyph_usage._space: introduce _space (utilize)(0)(); fi;
1938     if (name_to_code("space")<0) and (unknown code_to_name_32):
1939       encode("space") (32);
1940     fi
1941     beginglyph("_space") fix_hsbw(space,0,0); endglyph;
1942   fi
1943   if not name_used("nbspace"):
1944     if unknown glyph_usage._nbspace: introduce _nbspace (utilize)(0)(); fi;
1945     %
1946     beginglyph("_nbspace") fix_hsbw(space,0,0); endglyph; % normal space width
1947   fi
1948 fi
1949 endif;
1950 def complete_tfm_info =
1951 % complete fontdimen info:

```



```

1951 % |designsize| is expected to be known
1952 % |slant| dimen has already been set; |xheight| dimen – not necessarily,
1953 % but |pf_info_set.XHEIGHT| is known:
1954 completing_tfm_info:=1;
1955 pf_info_xheight whatever,
1956   if known lc_height#: lc_height# else: tfm_units(pf_info_set.XHEIGHT) fi;
1957 pf_info_normal_space space if known space#: , space# fi;
1958 pf_info_space_stretch space_stretch
1959   if known space_stretch#: , space_stretch# fi;
1960 pf_info_space_shrink space_shrink if known space_shrink#: , space_shrink# fi;
1961 pf_info_quad quad if known quad#: , quad# fi;
1962 pf_info_extra_space extra_space if known extra_space#: , extra_space# fi;
1963 font_math_rule math_rule;
1964 font_math_axis math_axis;
1965 % complete header info:
1966 pf_info_headerbyte 72, max(0, 254 - round 2designsize);
1967 completing_tfm_info:=whatever;
1968 enddef;
1969
1970 def BCPL_string(expr s,n)= % string |s| becomes an |n|-byte BCPL string
1971   for l:=if length(s)>=n: n-1 else: length(s) fi: l
1972     for k:=1 upto l: , substring (k-1,k) of s endfor
1973     for k:=l+2 upto n: , 0 endfor endfor
1974 enddef;
1975
1976 % The Old Tradition...
1977 def font_size expr x = designsize:=x enddef;
1978 def font_slant expr x = fontdimen 1: x enddef;
1979 def font_normal_space expr x = fontdimen 2: x enddef;
1980 def font_normal_stretch expr x = fontdimen 3: x enddef;
1981 def font_normal_shrink expr x = fontdimen 4: x enddef;
1982 def font_x_height expr x = fontdimen 5: x enddef;
1983 def font_quad expr x = fontdimen 6: x enddef;
1984 def font_extra_space expr x = fontdimen 7: x enddef;
1985
1986 % A New Tradition...
1987 def def_font_param (suffix param_name)(expr param_num, param_desc) =
1988   def param_name text x =
1989     begingroup save #; % cf. the definition of |tfm_units|
1990     if (known x#) or ((designsize<>0) and known x):
1991       pf_info_fontdimen param_num, tfm_units(x), "(" & param_desc & ")";
1992     fi
1993   endgroup
1994 enddef;
1995 enddef;
1996
1997 def_font_param (font_math_rule, 8, "math rule");
1998 def_font_param (font_math_axis, 22, "math axis");

```

```

1999 % symbol fonts
2000 def_font_param (font_num_one, 8, "num1");
2001 def_font_param (font_num_two, 9, "num2");
2002 def_font_param (font_num_three, 10, "num3");
2003 def_font_param (font_denom_one, 11, "denom1");
2004 def_font_param (font_denom_two, 12, "denom2");
2005 def_font_param (font_sup_one, 13, "sup1");
2006 def_font_param (font_sup_two, 14, "sup2");
2007 def_font_param (font_sup_three, 15, "sup3");
2008 def_font_param (font_sub_one, 16, "sub1");
2009 def_font_param (font_sub_two, 17, "sub2");
2010 def_font_param (font_sup_drop, 18, "sup_drop");
2011 def_font_param (font_sub_drop, 19, "sub_drop");
2012 def_font_param (font_delim_one, 20, "delim1");
2013 def_font_param (font_delim_two, 21, "delim2");
2014 % extension fonts
2015 def_font_param (font_big_op_spacing_one, 9, "big_op_spacing1");
2016 def_font_param (font_big_op_spacing_two, 10, "big_op_spacing2");
2017 def_font_param (font_big_op_spacing_three, 11, "big_op_spacing3");
2018 def_font_param (font_big_op_spacing_four, 12, "big_op_spacing4");
2019 def_font_param (font_big_op_spacing_five, 13, "big_op_spacing5");
2020
2021 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)=
```

FRAC

```

71 (frac.three.x2-frac.three.x1);
72 frac.three.x4-frac.three.x1=700;
73
74 frac.four.y1=0.08[frac.in.y1,frac.in.y2];
75 frac.four.y2=0.34[frac.in.y1,frac.in.y2];
76 frac.four.y3=0.51[frac.in.y1,frac.in.y2];
77 frac.four.y4=0.66[frac.in.y1,frac.in.y2];
78 frac.four.y5=0.92[frac.in.y1,frac.in.y2];
79
80 (frac.four.x1+frac.four.x5)/2=500;
81 (frac.four.x5-frac.four.x4)=
82 (frac.four.x4-frac.four.x3)=
83 (frac.four.x3-frac.four.x2)=
84 (frac.four.x2-frac.four.x1);
85 frac.four.x5-frac.four.x1=800;
86
87 frac.half.y1=0.10*latin_vcentre;
88 frac.half.y2=0.82*latin_vcentre;
89 frac.half.y3=latin_vcentre;
90 frac.half.y4=1.18*latin_vcentre;
91 frac.half.y5=1.90*latin_vcentre;
92
93 (frac.half.x1+frac.half.x2)/2=250;
94 frac.half.x2-frac.half.x1=330;
95
96 vardef hexdig(expr d) =
97   if d<10: decimal d else: char (d+87) fi
98 enddef;
99
100 vardef make_digit_set(expr xfm,thispage,place) =
101   numeric ccount;
102   ccount:=0;
103   forsuffices i=zero,one,two,three,four,five,six,seven,eight,nine:
104     begintsuglyph("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 Matthew Skala
4 %
5-29 [Standard copyright notice]
30
31 inclusion_lock(latinintro);
32
33 

---


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

```

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

```

```

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

```

```

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

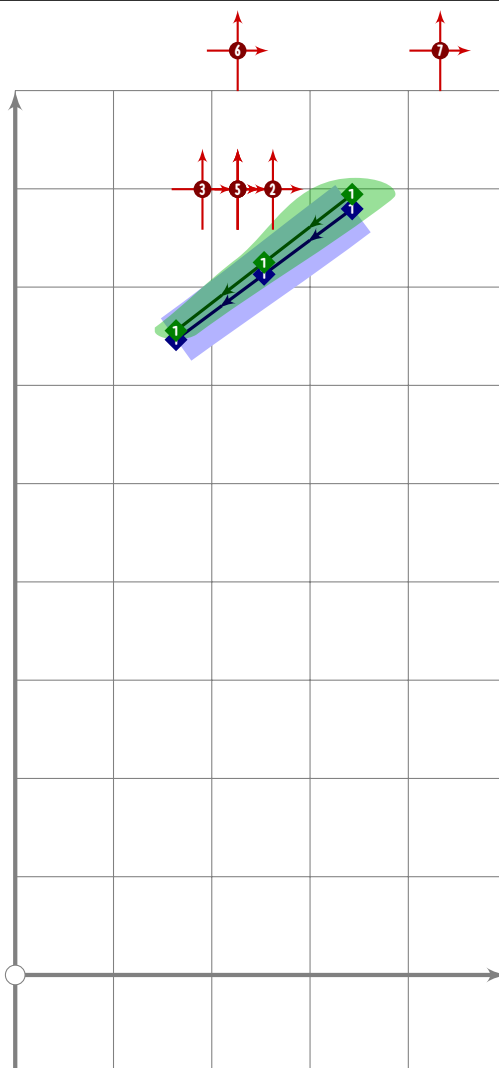
```

accent.mp

```

1 %
2 % Accents for Tsukurimashou
3 % Copyright (C) 2011, 2012, 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)-(1.9,1.9)-(1,1));
60   push_anchor(anc_upper,accent_default[anc_upper] shifted (0,150));
61   push_anchor(anc_grave,accent_default[anc_grave] shifted (0,150));
62   push_anchor(anc_acute,accent_default[anc_acute] shifted (0,150));
63   push_anchor(anc_wide,accent_default[anc_wide] shifted (0,150));
64   push_anchor(anc_tilde,accent_default[anc_tilde] shifted (0,150));
65   push_anchor(anc_ring,accent_default[anc_ring] shifted (0,150));
66   push_anchor(anc_caron_comma,
67             accent_default[anc_caron_comma] shifted (0,150));
68 endif;
69
70 vardef tsu_accent.caron =
71   push_anchor(-anc_wide,accent_default[anc_wide]);
72   push_stroke((500-1.5*tsu_punct_size,vmetric(0.95))-
73             (500,vmetric(0.80))-
74             (500+1.5*tsu_punct_size,vmetric(0.95)),
75             (2,2)-(1.4,1.4)-(1.4,1.4));
76   set_bosize(0,80);
77   set_botip(0,1,1);
78   push_anchor(anc_upper,accent_default[anc_upper] shifted (0,150));
79   push_anchor(anc_grave,accent_default[anc_grave] shifted (0,150));
80   push_anchor(anc_acute,accent_default[anc_acute] shifted (0,150));
81   push_anchor(anc_wide,accent_default[anc_wide] shifted (0,150));
82   push_anchor(anc_tilde,accent_default[anc_tilde] shifted (0,150));
83   push_anchor(anc_ring,accent_default[anc_ring] shifted (0,150));
84   push_anchor(anc_caron_comma,
85             accent_default[anc_caron_comma] shifted (0,150));
86 endif;
87
88 vardef tsu_accent.caron_comma =
89   push_anchor(-anc_caron_comma,accent_default[anc_caron_comma]);

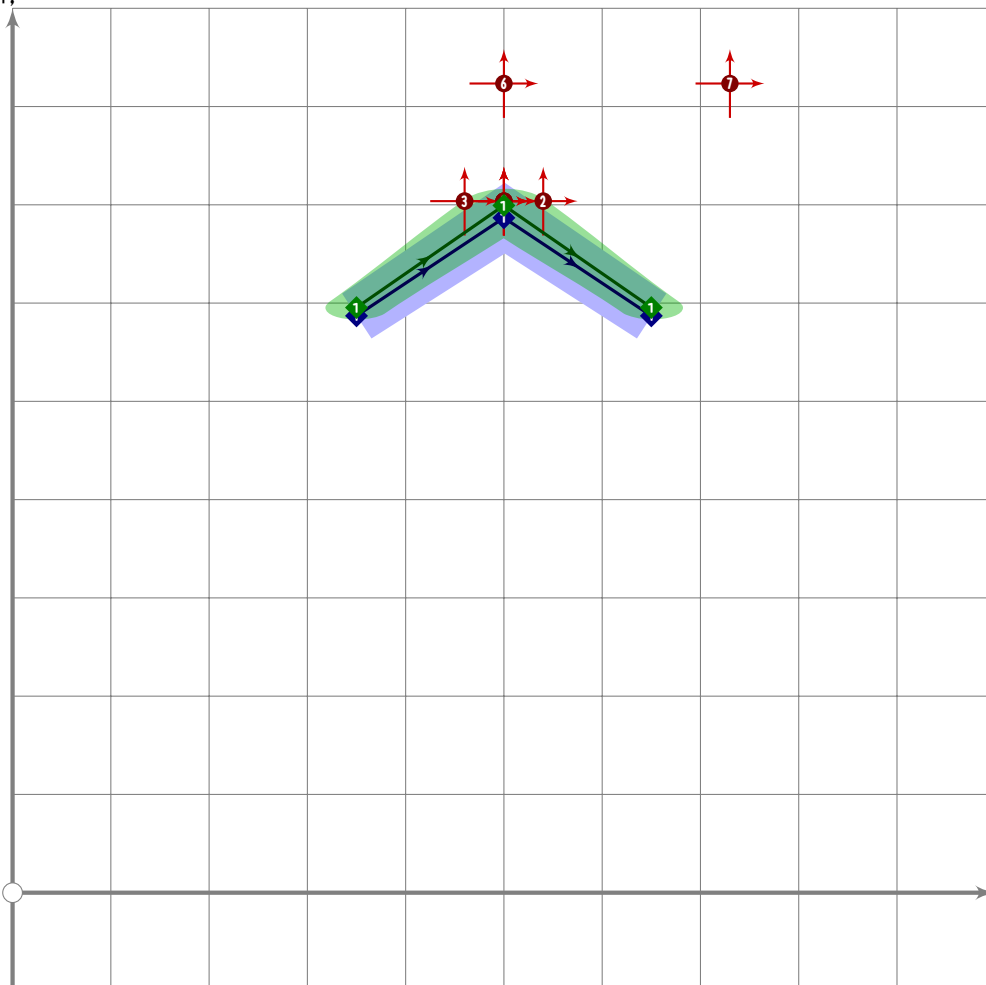
```


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(0,1,1);
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(0,1,1);

```

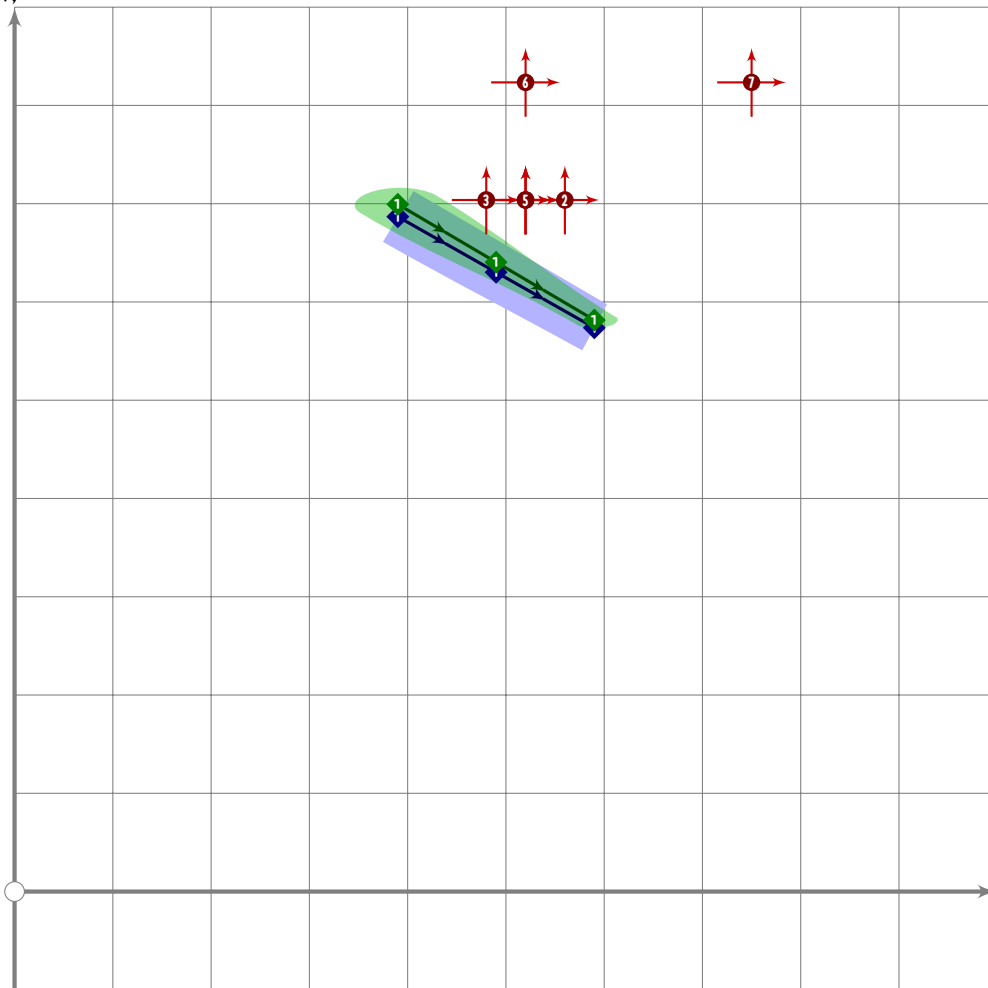
```

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 =

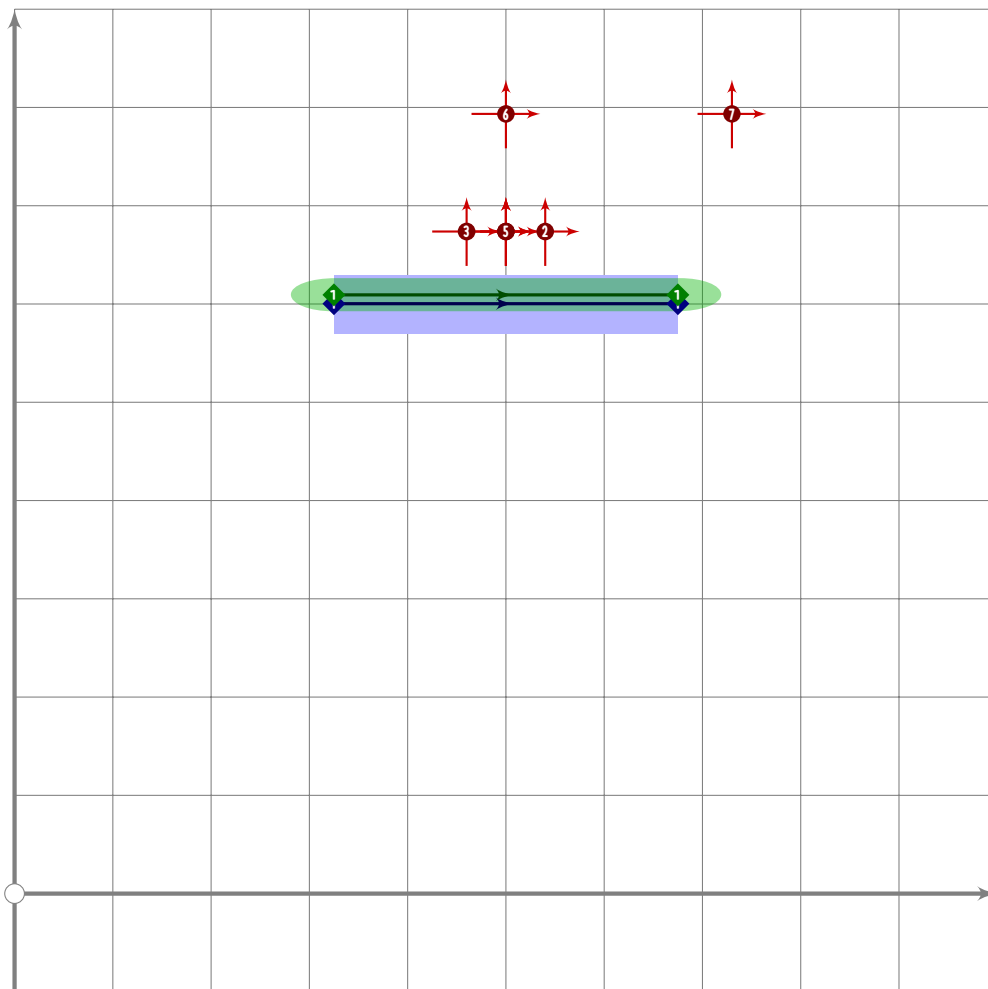
```

```

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(0,1,1);
209
210   push_stroke((500-0.4*tsu_punct_size,vmetric(0.95))-
211     (500-1.4*tsu_punct_size,vmetric(0.78)),
212     (2,2)-(1.6,1.6)-(1.3,1.3));
213   replace_strokep(0)(insert_nodes(oldp)(0.5));
214   set_bosize(0,80);
215   set_botip(0,1,1);
216   push_anchor(anc_upper,accent_default[anc_upper] shifted (0,180));
217   push_anchor(anc_grave,accent_default[anc_grave] shifted (0,180));
218   push_anchor(anc_acute,accent_default[anc_acute] shifted (0,180));
219   push_anchor(anc_wide,accent_default[anc_wide] shifted (0,180));
220   push_anchor(anc_tilde,accent_default[anc_tilde] shifted (0,180));
221   push_anchor(anc_ring,accent_default[anc_ring] shifted (0,180));
222   push_anchor(anc_caron_comma,
223     accent_default[anc_caron_comma] shifted (0,180));
224 enddef;

```

U+FFE3
tsuku.uniFFE3



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)

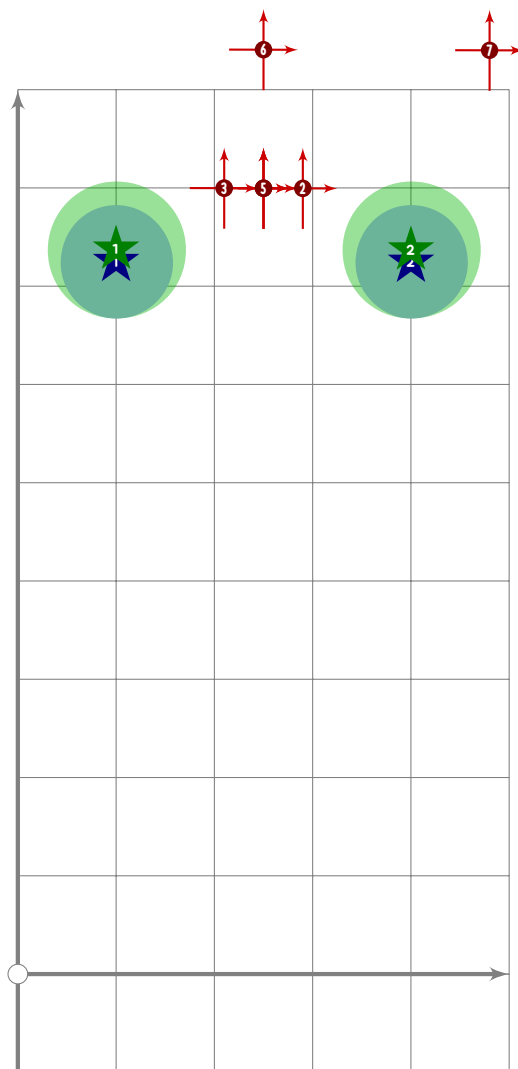
```

```

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}{-1.4,1}..(0,0)..(1.4,-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_bokeepshape(-1);
306   set_bokeepshape(0);
307   push_anchor(anc_upper,accent_default[anc_upper] shifted (0,150));
308   push_anchor(anc_grave,accent_default[anc_grave] shifted (0,150));
309   push_anchor(anc_acute,accent_default[anc_acute] shifted (0,150));
310   push_anchor(anc_wide,accent_default[anc_wide] shifted (0,150));
311   push_anchor(anc_tilde,accent_default[anc_tilde] shifted (0,150));
312   push_anchor(anc_ring,accent_default[anc_ring] shifted (0,150));

```

```

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

```



```

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

```

bcircle.mp

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

```

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

```

```
119 endgroup
120 enddef;
```

bkencl.mp

```
1 %
2 % Enclosure operations for building kanji
3 % Copyright (C) 2011, 2012, 2013, 2015, 2016 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.u9580.1_";
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 % note special calling convention - extra boolean for inclusion of "tick"
133 % seen in some Japanese and Korean characters
134 vardef build_kanji.long_stride_enclose(expr do_tick)(text contents) =
135   push_pbox_toexpand("build_kanji.long_stride_enclose");
136   perl_structure:=perl_structure&"eids.u2FFA.u5EF4._3";
137   begingroup
138     save myxf;
139     transform myxf;
140     (50,850) transformed myxf=(350,810);
141     (50,50) transformed myxf=(350,80);
142     (950,50) transformed myxf=(950,80);
143     tsu_xform(myxf)(contents);
144     save x,y;
145     numeric x[],y[];
146     x1=80;
147     x2=260;
148     x4=120;
149     y1=y2=780;
150     y4=460;
151     z3=0.4[z4,z2]+mincho*(30,0);
152     z5=(0.3-0.5*mincho)[z4,z2];
153     push_stroke(z1-z2..tension 1.2..z3..z5,
154       (1.6,1.6)-(1.6,1.6)-(1.6-0.1*mincho,1.6-0.1*mincho)-
155       (1.6-0.2*mincho,1.6-0.2*mincho));
156     set_boserif(0,1,4);
157     set_botip(0,1,0);
158     x7=0;
159     x9=300;
160     x10=220;
161     x11=100;
162     y7=y9=y4;
163     y10=80;
164     y11=-70;
165     z6=point 2.2 of get_strokep(0);
166     z8=(z7-z9) intersectionpoint (get_strokep(0)-((-1)[z4,z2]));

```

```

167   push_stroke(z6-z8-z9.tension 1.2..z10..z11,
168       (1.6-0.5*mincho,1.6-0.5*mincho)-(1.5-0.1*mincho,1.5-0.1*mincho)-
169       (1.6,1.6)-(1.4,1.4)-(1,1));
170   set_boserif(0,2,4);
171   set_botip(0,1,1);
172   set_botip(0,2,0);
173   if do_tick:
174       push_stroke((150,330)..(120,290)..(60,220),
175       (1.2,1.2)-(1.1,1.1)-(1.6,1.6));
176   fi;
177   push_stroke((130,300)..(390,30)..tension 1.6..(900,-50),
178       (1,1)-(1.6,1.6)-(1.9,1.9));
179   endgroup;
180   expand_pbox;
181   enddef;
182
183   vardef build_kanji.road_enclose(text contents) =
184       push_pbox_toexpand("build_kanji.road_enclose");
185       perl_structure:=perl_structure&"eids.u2FFA.u2ECC._3";
186       begingroup
187         save myxf;
188         transform myxf;
189         (50,850) transformed myxf=(315,850);
190         (50,-50) transformed myxf=(315,50);
191         (950,50) transformed myxf=(950,50);
192         tsu_xform(myxf)(contents);
193         push_stroke((100,770)..tension 1.2..(180,690)..(220,630),
194             (1,1)-(1.3,1.3)-(1.9,1.9));
195         set_bosize(0,92);
196         push_stroke((80,453)-(240,450)..(mincho[230,210],250)
197             ..tension 1.2..(mincho[180,140],50)..{curl 0.4}(60,-40),
198             (1.4,1.4)-(1.6,1.6)-(1.4,1.4)-(1.2,1.2)-(1,1));
199         set_botip(0,1,1);
200         set_botip(0,2,1);
201         set_boserif(0,1,4);
202         set_bosize(0,92);
203         push_stroke((point (2.3+mincho) of get_strokep(0))
204             {(1-2*mincho)*direction (2.3+mincho) of get_strokep(0)}..
205             (240,100)..(270,45+15*mincho)..(400,-10)..tension 3..(950,-20),
206             (1,1)-(1.1,1.1)-(1.1,1.1)-(1.7,1.7)-(1.9,1.9));
207         set_bosize(0,92);
208       endgroup;
209       expand_pbox;
210   enddef;
211
212   vardef build_kanji.steam_enclose(expr ur)(text contents) =
213       push_pbox_toexpand("build_kanji.steam_enclose");
214       begingroup

```



```

215 save xfa,xfb,xfc,xfd;
216 transform xfa,xfb,xfc,xfd;
217 (50,50) transformed xfc=(50,50);
218 (950,850) transformed xfc=ur;
219 xypart xfc=yxpart xfc=0;
220 tsu_xform(xfc)(contents);
221
222 (0,0) transformed xfa=(0,950) transformed xfc;
223 (1,1) transformed xfa=(280,810);
224 xypart xfa=yxpart xfa=0;
225 (0,0) transformed xfb=(1100,950) transformed xfc;
226 (1,1) transformed xfb=(970,810);
227 xypart xfb=yxpart xfb=0;
228 (0,1) transformed xfd=(1100,950) transformed xfc;
229 (1,0) transformed xfd=(1000,-50);
230 xypart xfd=yxpart xfd=0;
231
232 push_stroke(((1,1)..tension 1.2..(0.5,0.45)..(0,0.2)) transformed xfa,
233 (1.7,1.7)-(1.5,1.5)-(1.2,1.2));
234 set_boserif(0,0,10);
235 set_bosize(0,90);
236
237 push_stroke((get_strokep(0) intersectionpoint
238 (((0,0.8)-(1,0.8)) transformed xfa))-
239 ((0.5,0.8) transformed xfb),
240 (1.5,1.5)-(1.6,1.6));
241 set_boserif(0,1,9);
242 set_bosize(0,90);
243
244 push_stroke(((0.8,0.4) transformed xfa)-((0.15,0.4) transformed xfb),
245 (1.6,1.6)-(1.6,1.6));
246 set_boserif(0,1,9);
247 set_bosize(0,90);
248
249 push_stroke(((0.2,0) transformed xfa)-(interpath(mincho,
250 (0,1)..tension 1.6..(0.4,0)..(0.6,0)..tension 1.5..
251 (0.73,0.2)..(0.8,0.4),
252 (0,1)..tension 1.6..(0.25,0.2)..{right}(0.8,0){curl 1}..
253 (0.6,0.2)..(0.6,0.4)
254 ) transformed xfd),
255 (1.6,1.6)-(1.6,1.6)-(1.4,1.4)-
256 (1.4,1.4)-(1.2,1.2)-(0.9,0.9));
257 set_boserif(0,1,4);
258 set_botip(0,1,1);
259 set_bosize(0,90);
260 endgroup;
261 expand_pbox;
262 enddef;

```

```

263
264 vardef build_kanji.wind_enclose(text contents) =
265   push_pbox_toexpand("build_kanji.wind_enclose");
266   push_stroke((50,-50)..(120,100)..(160,300)..tension 1.2..(180,760)
267     -interpath(mincho,
268       (770,760){down}..{dir -72}(810,-20)..(860,-10)..tension 1.5..
269       (890,100)..(910,200),
270       (760,760){down}..(780,100)..{right}(910,-40){curl 1}..
271       (890,60)..(890,160)),
272     (1,1)-(1.3,1.3)-(1.5,1.5)-(1.6,1.6)-(1.6,1.6)-
273     (1.4,1.4)-(1.4,1.4)-(1.2,1.2)-(0.9,0.9));
274   set_botip(0,3,1);
275   set_botip(0,4,1);
276   set_boserif(0,3,4);
277   set_boserif(0,4,4);
278   begingroup
279     save t;
280     transform t;
281     (50,-50) transformed t=(230,-50);
282     (950,-50) transformed t=(700,-50);
283     (950,850) transformed t=(700,660+20*mincho);
284     tsu_xform(t)(contents);
285   endgroup;
286   expand_pbox;
287 enddef;

```

buildkanji.mp

```
1 %
2 % Build a kanji character by assembling parts
3 % Copyright (C) 2011, 2012, 2013, 2016 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);
```

```

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 1.1..(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)-(1.4,1.4)-(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_strokep(i) rotated -45;
136     for j=0 upto length pp:
137       x3:=xpart point j of pp;
138       y3:=ypart point j of pp;
139       if x3>x1:
140         x1:=x3;
141         y1:=y3;
142       fi;
143       if y3>y2:
144         x2:=x3;
145         y2:=y3;
146       fi;
147     endfor;
148     i:=i-1;
149   endfor;
150   (0,0) transformed myxf=(z2 rotated 45);
151   (1,8,0) transformed myxf=(z1 rotated 45);
152   xpart myxf=0;
153   ypart ((0,1) transformed myxf)=830;
154   push_stroke(
155     ((0.5,0.9)..tension 1.2..(0,0)..(-0.5,-0.5)) transformed myxf,
156     (1.7,1.7)-(1.3,1.3)-(1,1));
157   set_boserif(0,0,10);
158   push_stroke(
159     ((0.4,0.65)-(1.3,0.65)..tension 1.2..(1.14,0.3)..(0.88,0))
160     transformed myxf,
161     (1.6,1.6)-(1.6,1.6)-(1.4,1.4)-(1,1));
162   set_boserif(0,1,4);
163   set_botip(0,1,0);
164 endgroup;
165 perl_structure:=perl_structure&"]";
166 enddef;

```

BUIL

```

167
168 vardef build_kanji.attach_tick(expr newtop)(text curves) =
169   perl_structure:=
170     perl_structure&"['build_kanji.attach_fishhook",eids.u2FF1.u31D2._1,[';
171   begingroup
172     save atosp,atnsp;
173     numeric atosp,atnsp;
174     atosp:=sp;
175     curves;
176     perl_structure:=perl_structure&""];
177     atnsp:=sp;
178     save i,thisy,maxy;
179     maxy:=-infinity;
180     i:=0;
181     forever:
182       exitif find_stroke(i)<atosp;
183       if (xpart (get_stroke(i) intersectiontimes
184         ((400,900)-(500,200)-(600,900))))>=0:
185         thisy:=ypart urcorner get_stroke(i);
186         if thisy>maxy:
187           maxy:=thisy;
188         fi;
189       fi;
190       i:=i+1;
191     endfor;
192     sp:=atosp;
193     tsu_xform(identity yscaled (newtop/maxy))(sp:=atnsp);
194     push_stroke((500,190+newtop)-(440,newtop),(1.7,1.7)-(1,1));
195     set_boserif(0,0,10);
196   endgroup;
197   perl_structure:=perl_structure&""];
198 enddef;
199
200 vardef hook.box_bottom(expr sides_i,bottom_i) =
201   if (obstacktype[sides_i]=otstroke) and (obstacktype[bottom_i]=otstroke):
202     if (3=length obstackp[sides_i]) and (1=length obstackp[bottom_i]):
203       begingroup;
204       save x,y,p,e;
205       numeric x[],y[],e[];
206       path p[];
207
208       p1=obstackp[sides_i];
209       p2=obstackp[bottom_i];
210
211       z1=-(direction 0 of p1)/abs(direction 0 of p1);
212       z2=(direction 0.5 of p2)/abs(direction 0.5 of p2);
213       z3=(direction 3 of p1)/abs(direction 3 of p1);
214

```

BUIL

```

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 enddef;
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 enddef;

```

```

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.go_enclose(expr w,o)(text inside) =
277   push_pbox_toexpand("build_kanji.go_enclose");
278   build_kanji.lcr(480-w/2-20,o)(480+w/2+20,o)
279     (build_kanji.harmonic(240,0.86,50)(kanji.leftrad.person))
280     (inside)
281     (build_kanji.harmonic(240,0.86,50)(kanji.grthree.thumbtack));
282   expand_pbox;
283 enddef;
284
285 vardef build_kanji.harmonic(expr gap,sval,lspread)(text curves) =
286   perl_structure:=perl_structure
287     &"[build_kanji.harmonic,eids.u2FF1.u003F._1,['";
288   begingroup
289     save myxf;
290     transform myxf;
291     (50,50) transformed myxf=(50,50);
292     (950,50) transformed myxf=(950,50);
293     (50,850) transformed myxf=(50,850-gap);
294     tsu_xform(myxf)(curves);
295     perl_structure:=perl_structure&"],[";
296     save hsp;
297     hsp:=sp;
298     tsu_xform(myxf)(build_kanji.spread_legs(lspread)(curves));
299     save itp,toclip,nadded;
300     numeric itp,toclip,nadded;
301     path tp;
302     i:=0;
303     toclip:=0;
304     nadded:=0;
305     forever:
306       tp:=get_strokep(i);
307       if (ypart ulcorner tp<450)
308         or (abs(ypart (direction 0 of tp)/abs(direction 0 of tp))>0.95)
309         or (abs(ypart (direction infinity of tp)/
310           abs(direction infinity of tp))>0.95):

```

BUIL


```

311         set_bosize(i,0);
312         toclip:=toclip+1;
313     else:
314         replace_strokep(i)
315             (tp shifted -(0.5[ulcorner tp,lrcorner tp])
316                 scaled sval
317                 shifted ((0,gap)+0.5[ulcorner tp,lrcorner tp]));
318         set_bosize(i)(get_bosize(i)*sqrt(sval));
319         toclip:=toclip+2;
320         nadded:=nadded+1;
321     fi;
322     i:=i-1;
323     exitif find__stroke(i)<hsp;
324 endfor;
325 endgroup;
326 perl_structure:=perl_structure&"]]";
327 endif;
328
329 vardef build_kanji.lcr(expr splitpointa,overlapa)(expr splitpointb,overlapb)
330 (text leftstuff)(text centrestuff)(text rightstuff) =
331 perl_structure:=perl_structure
332 &"["build_kanji.lcr;eids.u2ff2._2.1_1.2_[";
333 begingroup
334     save t;
335     transform t[];
336     yypart t1=yypart t2=yypart t3=1;
337     ypart t1=yxpart t1=xy part t1=0;
338     ypart t2=yxpart t2=xy part t2=0;
339     ypart t3=yxpart t3=xy part t3=0;
340     (50,0) transformed t1=(50,0);
341     (950,0) transformed t1=(splitpointa+overlapa/2,0);
342     (50,0) transformed t2=(splitpointa-overlapa/2,0);
343     (950,0) transformed t2=(splitpointb+overlapb/2,0);
344     (50,0) transformed t3=(splitpointb-overlapb/2,0);
345     (950,0) transformed t3=(950,0);
346     tsu_xform(t1)(leftstuff);
347     perl_structure:=perl_structure&"],[";
348     tsu_xform(t2)(centrestuff);
349     perl_structure:=perl_structure&"],[";
350     tsu_xform(t3)(rightstuff);
351 endgroup;
352 perl_structure:=perl_structure&"]]";
353 endif;
354
355 vardef build_kanji.lean_to(expr lr) =
356 push_stroke((120,620)-(880,620),(1.6,1.6)-(1.6,1.6));
357 set_boserif(0,1,9);
358 begingroup

```

```

359   save ltxf;
360   transform ltxf;
361   xypart ltxf=yxpart ltxf=0;
362   (60,800) transformed ltxf=(60,800);
363   (360,30) transformed ltxf=lr;
364   push_stroke(
365     ((360,800)..tension 1.2..(270,300)..(60,30)) transformed ltxf,
366     (1.6,1.6)-(1.4,1.4)-(0.9,0.9));
367   endgroup;
368   set__boserif(0,0,10);
369 enddef;
370
371 vardef build_kanji.level(text curves) =
372   begingroup
373     save xsp;
374     xsp:=sp;
375     curves;
376     save lsum,denom,i;
377     lsum:=0;
378     denom:=0;
379     i:=0;
380     forever:
381       exitif find_stroke(i)<xsp;
382       if unknown get__bosize(i):
383         set__bosize(i,100);
384       fi;
385       if (get__bosize(i)>0):
386         lsum:=lsum+mlog(get__bosize(i));
387         denom:=denom+1;
388       fi;
389       i:=i+1;
390     endfor;
391     i:=0;
392     forever:
393       exitif find_stroke(i)<xsp;
394       if get__bosize(i)>0:
395         set__bosize(i,mexp(lsum/denom));
396       fi;
397       i:=i+1;
398     endfor;
399   endgroup;
400 enddef;
401
402 vardef build_kanji.lstransform(expr thresh,dist,mypt) =
403   if ypart mypt>thresh:
404     mypt
405   else:
406     (xpart mypt,

```

BUIL

```

407      (ypart mypt)*((thresh-dist)/thresh)
408      +(xpart mypt/1000)[(dist/thresh)*ypart mypt,dist]]
409  fi
410 enddef;
411
412 vardef build_kanji.lift_skirt(expr thresh,dist)(text curves) =
413   begingroup
414     save osp;
415     numeric osp;
416     osp:=sp;
417     curves;
418     save i,boti,x,y;
419     numeric x[],y[];
420     i:=0;
421     boti:=whatever;
422     y0:=1000;
423     forever:
424       exitif find_stroke(i)<osp;
425       if (ypart llcorner get_stroke(i))=(ypart urcorner get_stroke(i)):
426         if (unknown boti) or (ypart llcorner get_stroke(i)<y0):
427           y0:=ypart llcorner get_stroke(i);
428           boti:=i;
429         fi;
430       fi;
431       replace_stroke(i)(
432         for j=0 upto length oldp-1:
433           build_kanji.lstransform(thresh,dist)(point j of oldp)
434           ..controls build_kanji.lstransform(thresh,dist)(postcontrol j of oldp)
435           and build_kanji.lstransform(thresh,dist)(precontrol j+1 of oldp)..
436         endfor
437         if cycle oldp:
438           cycle
439         else:
440           build_kanji.lstransform(thresh,dist)(point infinity of oldp)
441         fi);
442       i:=i-1;
443     endfor;
444     if (known boti) and (y0<=thresh):
445       replace_stroke(boti)
446       (((0,7)+point 0 of oldp)..tension 1.2..((0,-5)+point 0.5 of oldp)..
447       ((0,10)+point 1 of oldp));
448       replace_strokeq(boti)((1.7,1.7)-(1.5,1.5)-(1,1));
449       set_boserif(boti,1,whatever);
450     fi;
451   endgroup;
452 enddef;
453
454 vardef build_kanji.lr(expr splitpoint,overlap)

```

```

455 (text leftstuff)(text rightstuff) =
456 perl_structure:=perl_structure
457   &"['build_kanji.lr",eids.u2ff0._11_',[";
458 begingroup
459   save t;
460   transform t[];
461   yypart t1=yypart t2=1;
462   ypart t1=yxpart t1=xy part t1=y part t2=yxpart t2=xy part t2=0;
463   (50,0) transformed t1=(50,0);
464   (950,0) transformed t1=(splitpoint+overlap/2,0);
465   (50,0) transformed t2=(splitpoint-overlap/2,0);
466   (950,0) transformed t2=(950,0);
467   tsu_xform(t1)(leftstuff);
468   perl_structure:=perl_structure&"',[";
469   tsu_xform(t2)(rightstuff);
470 endgroup;
471 perl_structure:=perl_structure&"']";
472 endif;
473
474 vardef build_kanji.sscale(text tran)(text curves) =
475   tsu_xform(identity shifted (-centre_pt) tran shifted centre_pt)(curves);
476 endif;
477
478 vardef build_kanji.spread_legs(expr dist)(text curves) =
479   begingroup
480     save osp;
481     numeric osp;
482     osp:=sp;
483     curves;
484     save mytr;
485     transform mytr[];
486     (50,50) transformed mytr1=(50,50);
487     (500,50) transformed mytr1=(500-dist/2,50);
488     (500,850) transformed mytr1=(500-dist/2,850);
489     (950,50) transformed mytr2=(950,50);
490     (500,50) transformed mytr2=(500+dist/2,50);
491     (500,850) transformed mytr2=(500+dist/2,850);
492     save i;
493     i:=0;
494     forever:
495       exitif find_stroke(i)<osp;
496       if xpart 0.75[llcorner get_stroke(i),urcorner get_stroke(i)]<475:
497         replace_stroke(i,oldp transformed mytr1);
498       elseif xpart 0.25[llcorner get_stroke(i),urcorner get_stroke(i)]>525:
499         replace_stroke(i,oldp transformed mytr2);
500       fi;
501       i:=i+1;
502     endfor;

```

```

503 endgroup;
504 enddef;
505
506 vardef build_kanji.tb(expr splitpoint,overlap)
507 (text topstuff)(text bottomstuff) =
508 perl_structure:=perl_structure
509 &"['build_kanji.tb',eids.u2ff1._1.1,'"];
510 begingroup
511 save t;
512 transform t[];
513 xpart t1=xxpart t2=1;
514 xpart t1=xy part t1=yxpart t1=xpart t2=xy part t2=yxpart t2=0;
515 (0,850) transformed t1=(0,850);
516 (0,-50) transformed t1=(0,splitpoint-overlap/2);
517 (0,850) transformed t2=(0,splitpoint+overlap/2);
518 (0,-50) transformed t2=(0,-50);
519 tsu_xform(t1)(topstuff);
520 perl_structure:=perl_structure&"],[";
521 tsu_xform(t2)(bottomstuff);
522 endgroup;
523 perl_structure:=perl_structure&""]";
524 enddef;
525
526 vardef build_kanji.tcb(expr splitpointa,overlapa)(expr splitpointb,overlapb)
527 (text topstuff)(text centrestuff)(text bottomstuff) =
528 perl_structure:=perl_structure
529 &"['build_kanji.tcb',eids.u2ff3._2.1_1.2,'"];
530 begingroup
531 save t;
532 transform t[];
533 xpart t1=xxpart t2=xxpart t3=1;
534 xpart t1=xy part t1=yxpart t1=0;
535 xpart t2=xy part t2=yxpart t2=0;
536 xpart t3=xy part t3=yxpart t3=0;
537 (0,850) transformed t1=(0,850);
538 (0,-50) transformed t1=(0,splitpointa-overlapa/2);
539 (0,850) transformed t2=(0,splitpointa+overlapa/2);
540 (0,-50) transformed t2=(0,splitpointb-overlapb/2);
541 (0,850) transformed t3=(0,splitpointb+overlapb/2);
542 (0,-50) transformed t3=(0,-50);
543 tsu_xform(t1)(topstuff);
544 perl_structure:=perl_structure&"],[";
545 tsu_xform(t2)(centrestuff);
546 perl_structure:=perl_structure&"],[";
547 tsu_xform(t3)(bottomstuff);
548 endgroup;
549 perl_structure:=perl_structure&""]";
550 enddef;

```

```

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

```

BUIL

```

599   curves;
600   save i;
601   i:=0;
602   forever:
603     exitif find_stroke(i)<osp;
604     replace_strokep(i)(
605       for j=0 upto length oldp-1:
606         build_kanji.wptransform(ca,cb,cc,cd)(point j of oldp)
607         ..controls build_kanji.wptransform(ca,cb,cc,cd)(postcontrol j of oldp)
608         and build_kanji.wptransform(ca,cb,cc,cd)(precontrol j+1 of oldp)..
609       endfor
610       if cycle oldp:
611         cycle
612       else:
613         build_kanji.wptransform(ca,cb,cc,cd)(point infinity of oldp)
614       fi);
615     i:=i-1;
616   endfor;
617 endgroup;
618 enddef;

```

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)..(50,30)..(100,-30)) transformed xf,
40     (1,1)..(1.4,1.4)..(1.8,1.8));
41   set_bosize(0.85);
42   set_boserif(0.2,4);
43
44   push_stroke(((0,80)..(50,30)..(100,-30)) transformed xf,
45     (1,1)..(1.4,1.4)..(1.8,1.8));
46   set_bosize(0.85);
47   set_boserif(0.2,4);
48 enddef;
49
50 vardef handakuten(expr location) =
51   push_lcblob(fullcircle scaled handakuten_outer shifted location
52     transformed inverse tsu_rescale_xform);
53   push_lcblob(reverse fullcircle scaled handakuten_inner shifted location
54     transformed inverse tsu_rescale_xform);
55 enddef;
```

DAKU

enclosed.mp

```
1 %
2 % Enclosed characters for Tsukurimashou
3 % Copyright (C) 2011, 2012, 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(0,1,1);
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 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:
```

GENJ

```

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

```

```

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

```

```

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

```

```

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

```

```

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

```



```

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

```

```

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

```

```

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

```

```

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

```

```

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

```

```

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

```

hiragana.mp

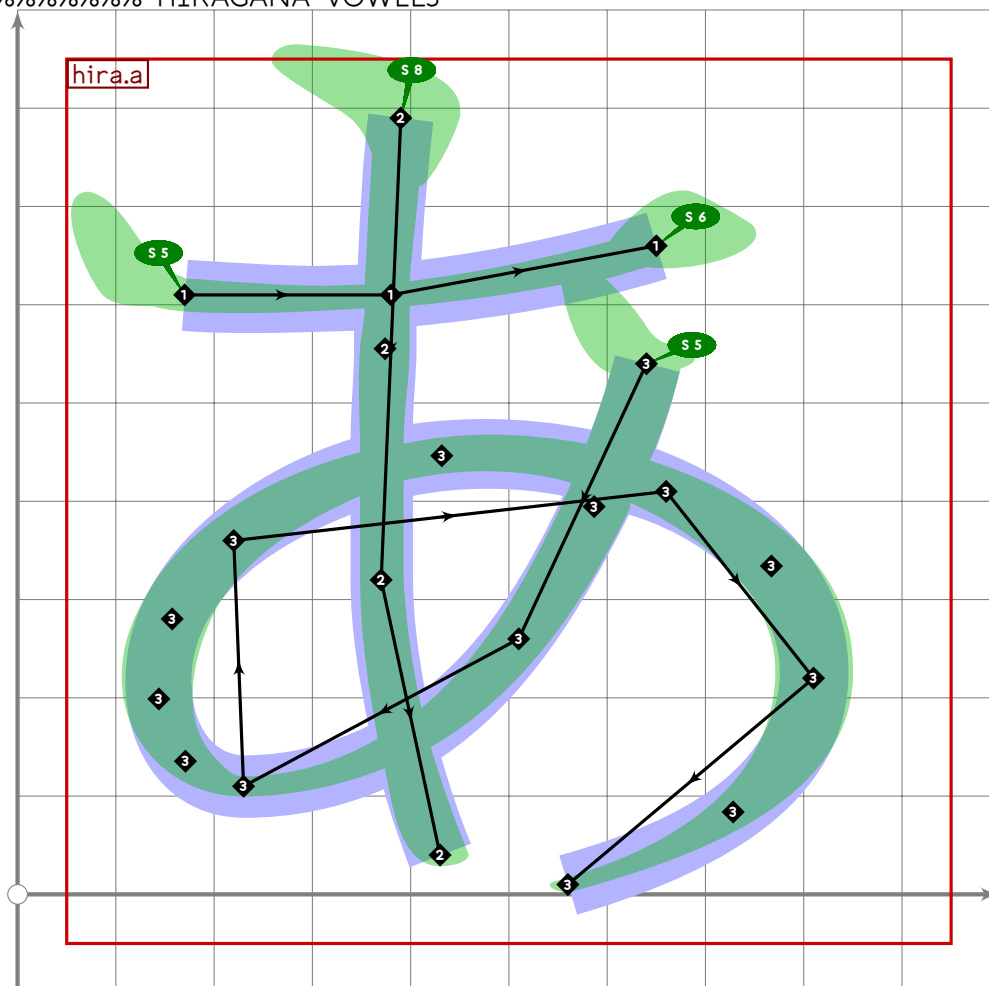
```

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

```

Hiragana Vowels

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



HIRA

```

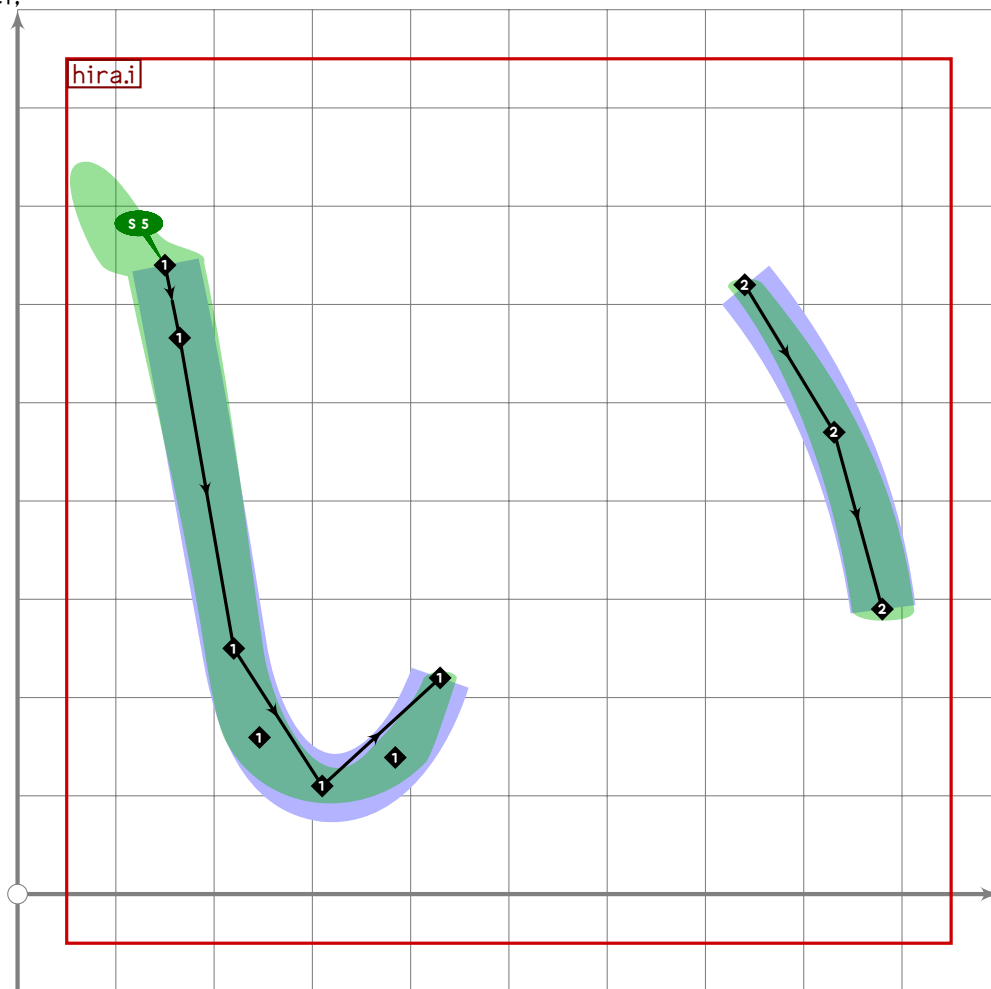
36
37 vardef hira.a =
38   push_pbox_toexpand("hira.a");
39
40   push_stroke((170,610)..(380,610)..(650,660),
41     (1,6,1,7)-(1,4,1,4)-(1,6,1,6));

```

```

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

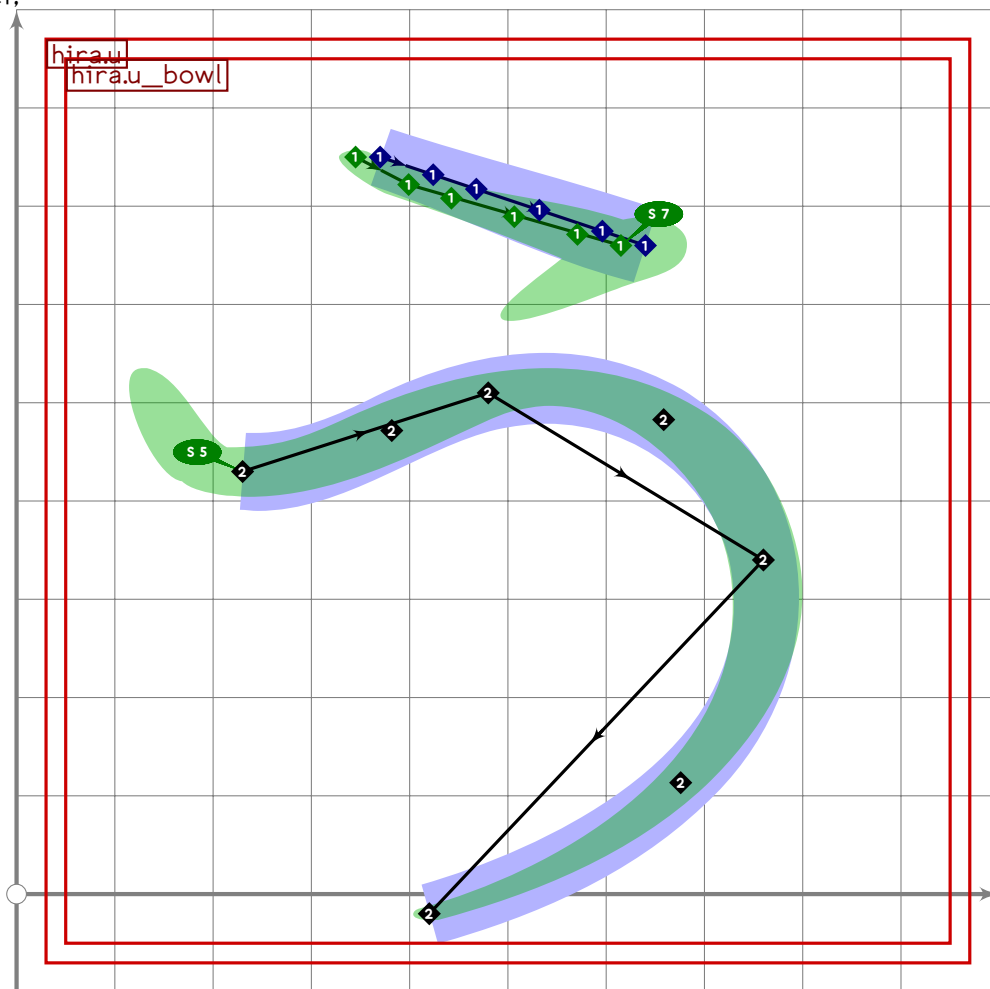
```



```

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

```



HIRA

```

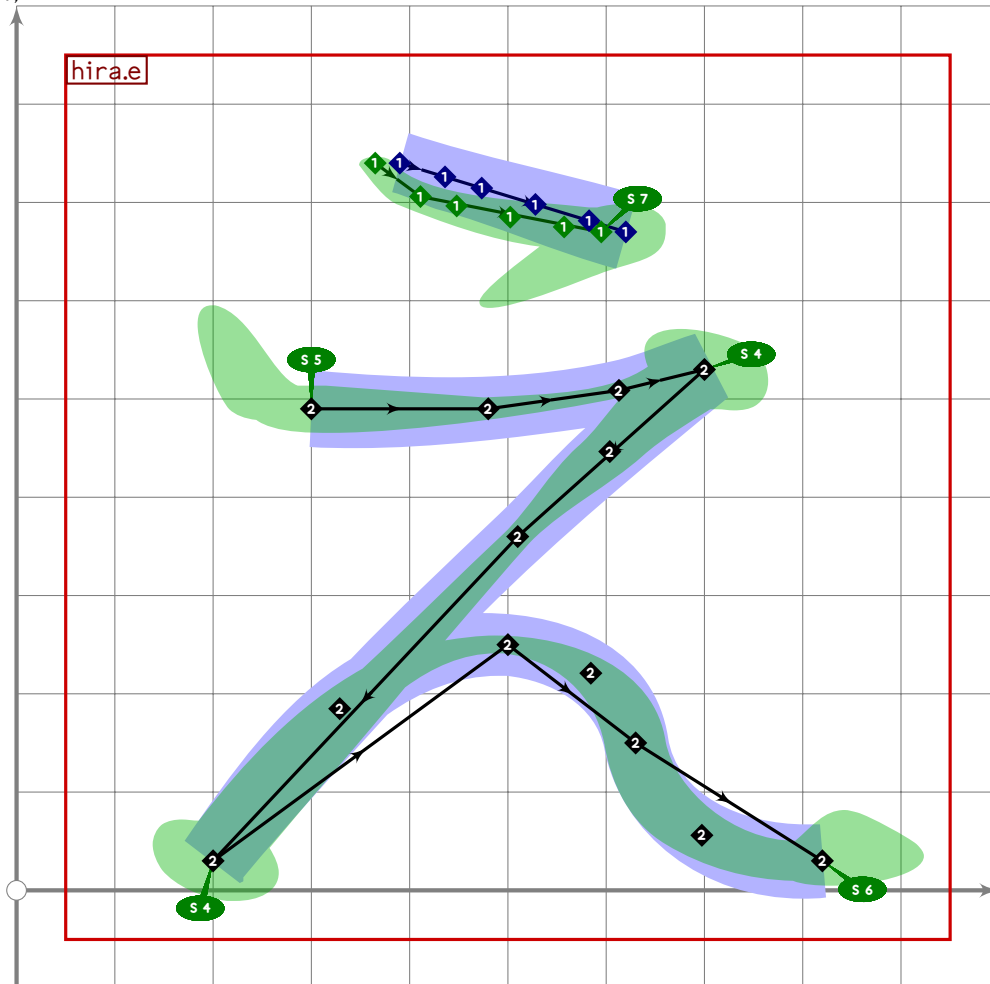
77
78 vardef hira.u =
79  push_pbox_toexpand("hira.u");
80
81  push_stroke(((370,750)..(0.2[(370,750),(640,660)]+10*down*mincho)..
82    tension 2..(640,660)) shifted (25*left*mincho),
83    (1,1)..(1.4,1.4)..(2.3,2.3));

```

```

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

```



```

89
90 vardef hira.e =
91   push_pbox_toexpand("hira.e");
92
93   push_stroke(((390,740)..(0.2[(390,740),(620,670)]+20*down*mincho)..
94     tension 2..(620,670)) shifted (25*left*mincho),
95     (1,1)..(14,14)..(2,3,2,3));
96   set_boserif(0,2,7);
97
98   push_stroke((300,490)..(480,490)..{curl 1}(700,530){curl 1}..
99     (510,360)..{curl 1}(200,30){curl 0}..
100     (500,250)..(630,150){dir 280}..{dir 5}(820,30),
101     (2,2,2,2)-(1,3,1,3)-(1,1)-(2,0,1,2,0,1)
102     -(1,1)-(1,7,1,7)
103     -(1,2,1,2)-(1,4,1,4)-(2,2));
104   replace_stroke(0)(insert_nodes(oldp)(1,6));

```

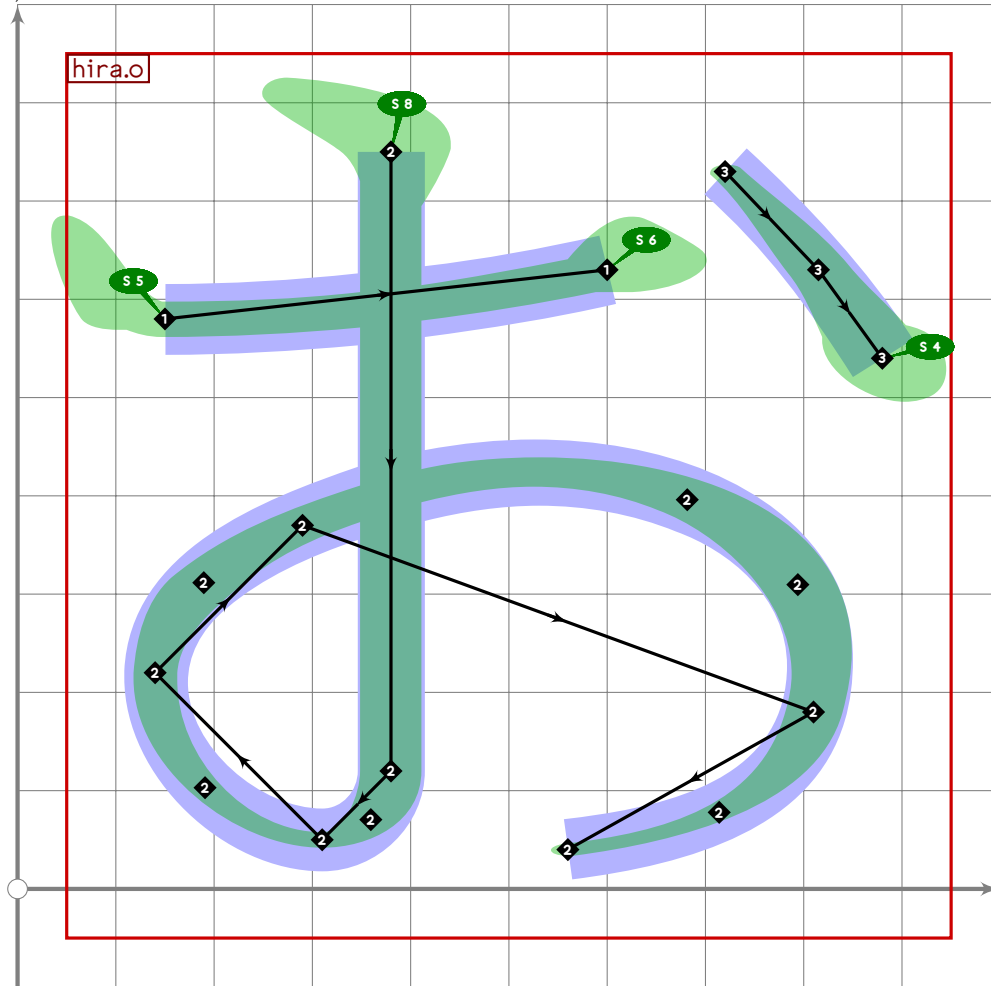
HIRA

U+304A
tsuku.uni304A

```

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

```



HIRA

```

113
114 vardef hira.o =
115   push_pbox_toexpand("hira.o");
116
117   push_stroke((150,580){right}..(600,630),
118     (1,8,1,8)..(1,8,1,8));
119   set_boserif(0,0,5);
120   set_boserif(0,1,6);
121
122   push_stroke((380,750)..(380,120){down}..(310,50){left}.tension 11..
123     (140,220)..(290,370)..(810,180)..{curl 0}(560,40),
124     (1,4,1,4)..(1,3,1,3)..(1,1)..(1,5,1,5)..(1,6,1,6)..
125     (1,6,1,6)..(1,1));

```

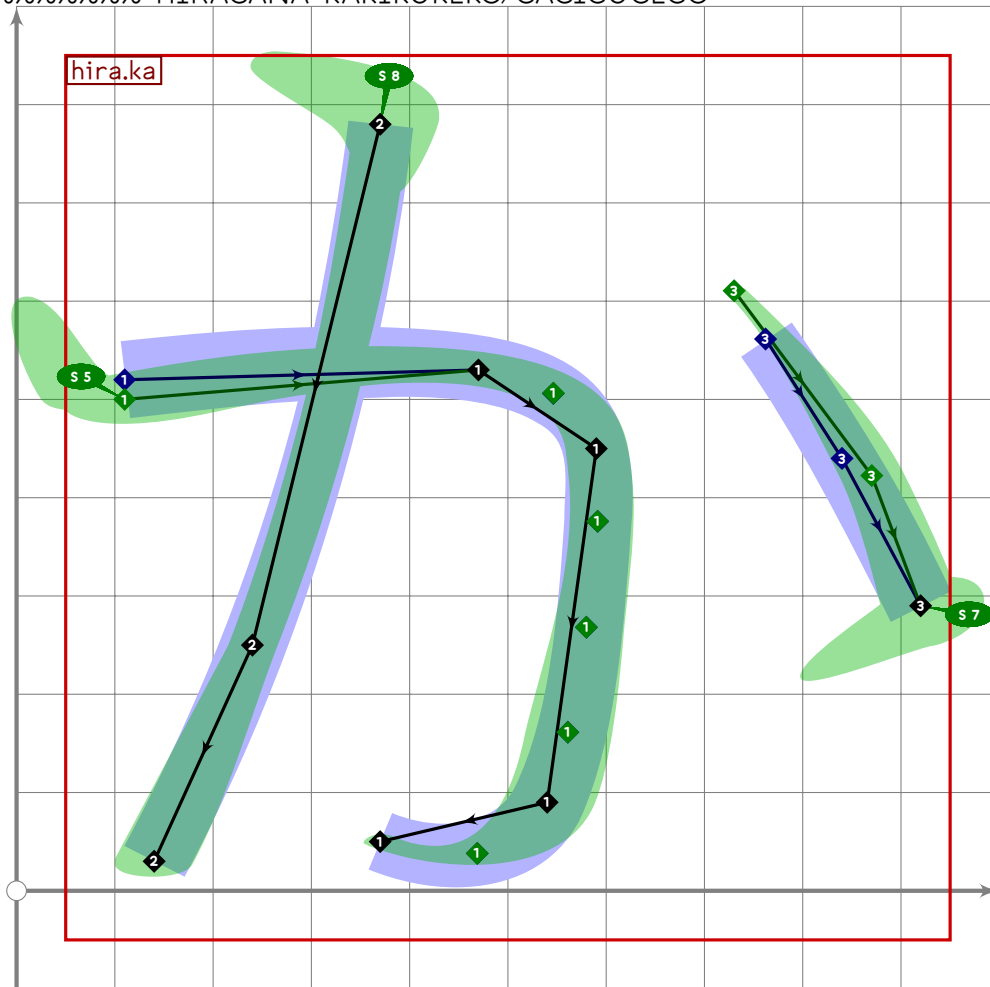
```

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

```

Hiragana Kakikukeko/Gagigugego

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



HIRA

```

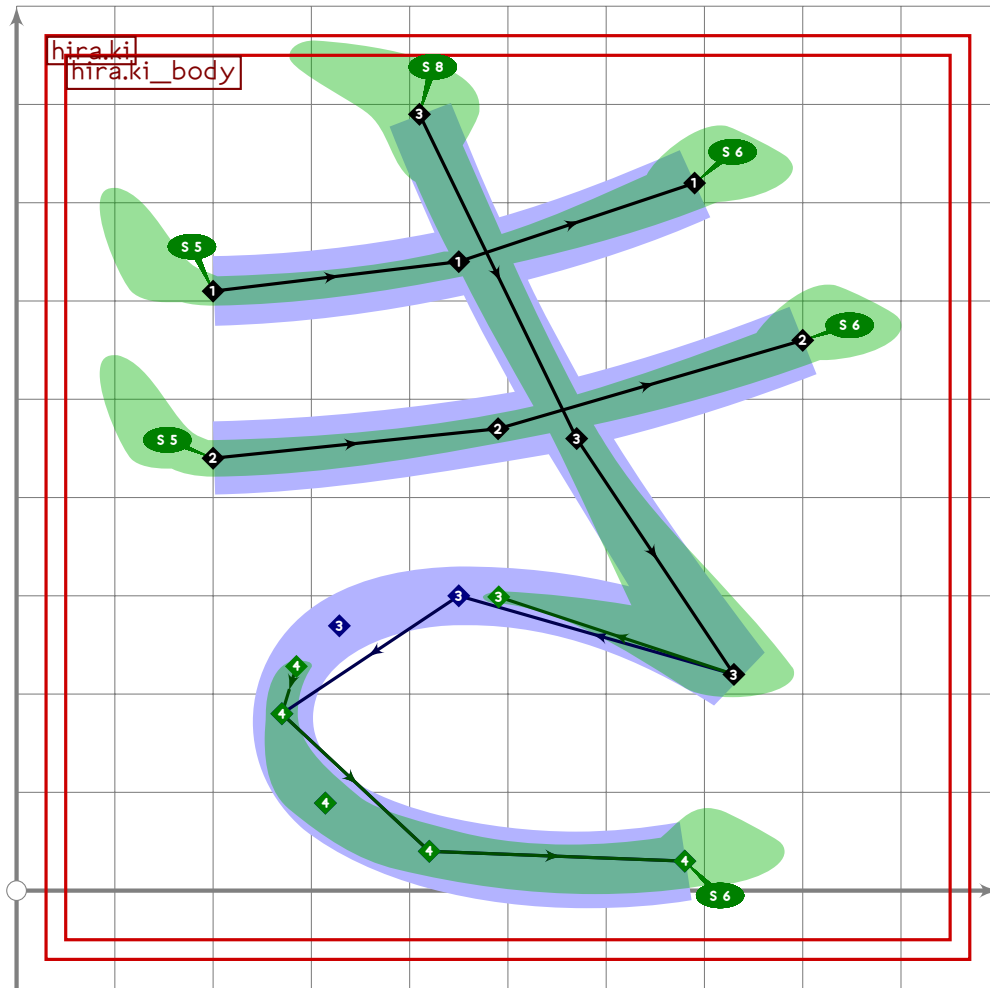
135
136 vardef hira.ka =
137   push_pbox_toexpand("hira.ka");
138
139   push_stroke(((110,520)+20*mincho*down){curl 0}..(470,530)..(590,450)..
140     tension 2..(540,90)..{curl 0.3}(370,50),
141     (2.3,2.3)..(1.7,1.7)..(14,14)..(18,18)..(1,1));
142   set_boserif(0,0,5);
143

```

```

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

```

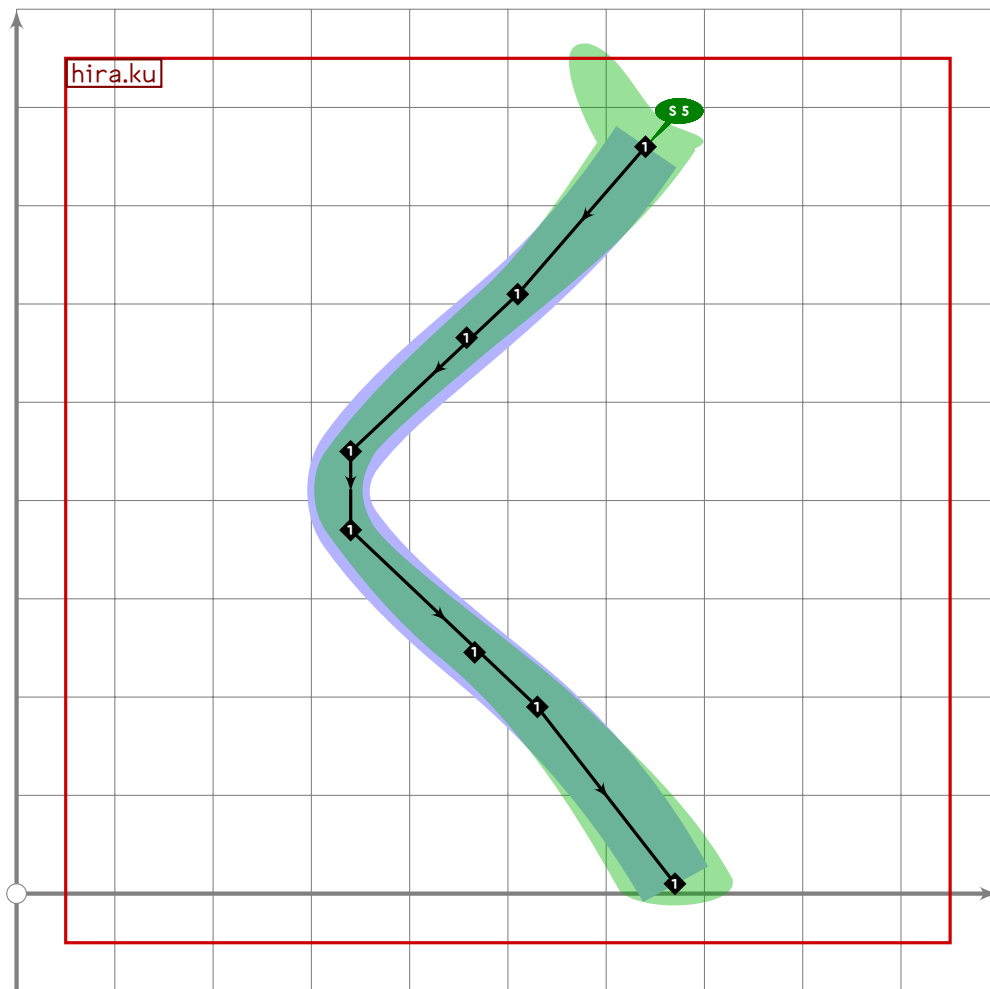


```

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

```

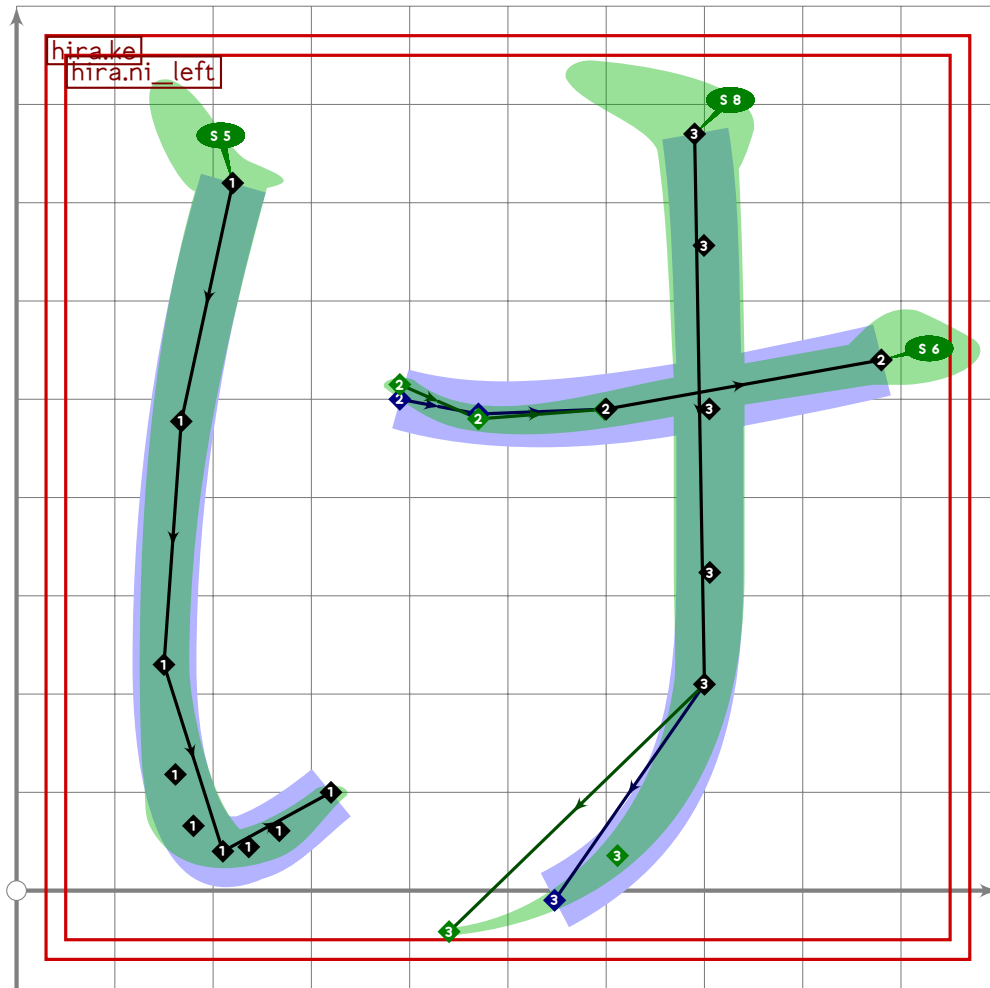
HIRA



```

183
184 vardef hira.ku =
185   push_pbox_toexpand("hira.ku");
186
187   push_stroke((640,760)..(510,610)..(340,450)..
188     tension 0.75..(340,370)..(530,190)..(670,10),
189     (1.9,1.9)..(1.6,1.6)..(1.2,1.2)..(1.2,1.2)..(1.7,1.7)..(2.1,2.1));
190   set_boserif(0,0,5);
191   expand_pbox;
192 enddef;

```

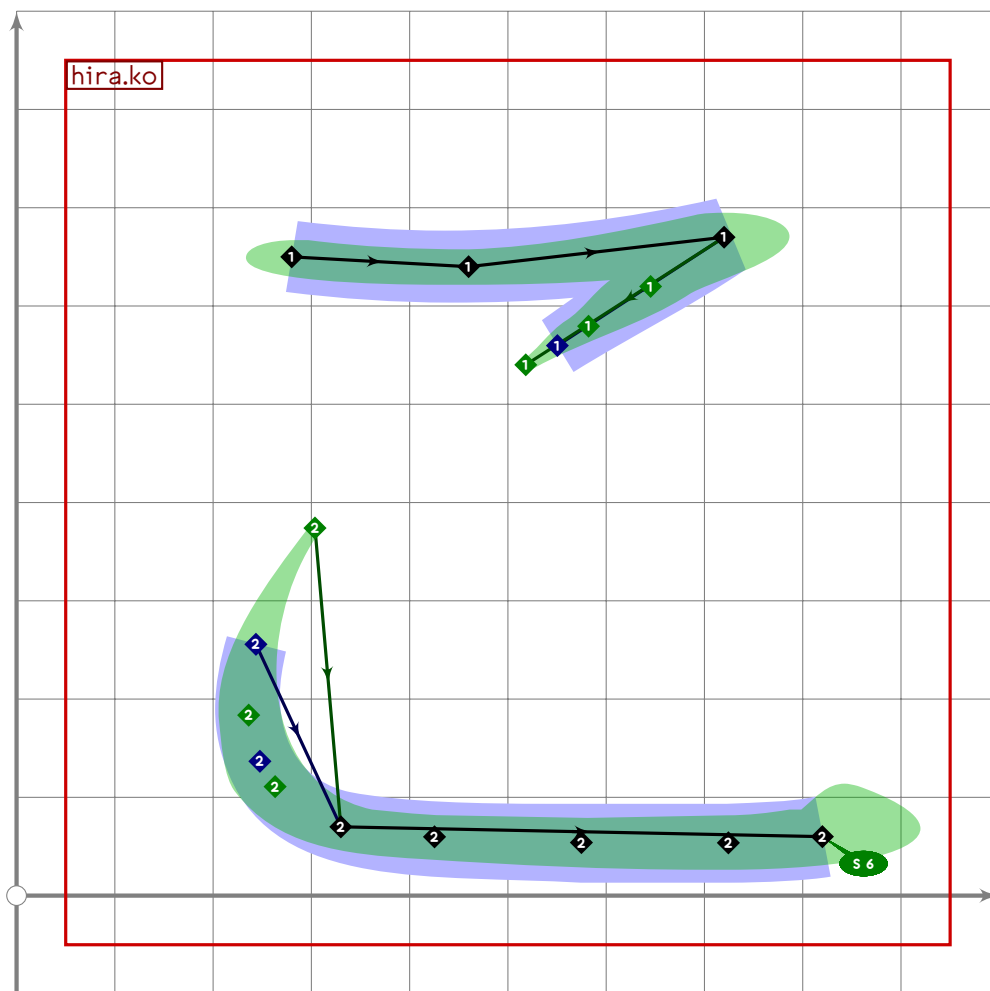


```

193
194 vardef hira.ke =
195   push_pbox_toexpand("hira.ke");
196
197   hira.ni_left;
198
199   push_stroke((390,500+15*mincho)..(470,485-5*mincho)..(600,490)..(880,540),
200     (1,1)..(14,14)..(18,18)..(2,2));
201   set_boserif(0,3,6);
202
203   push_stroke((690,770)..tension 2..(700,210)..(280,-10),
204     (1.6,1.6)-(14,14)-(0.6,0.6));
205   set_boserif(0,0,8);
206   expand_pbox;
207 enddef;

```

HIRA



```

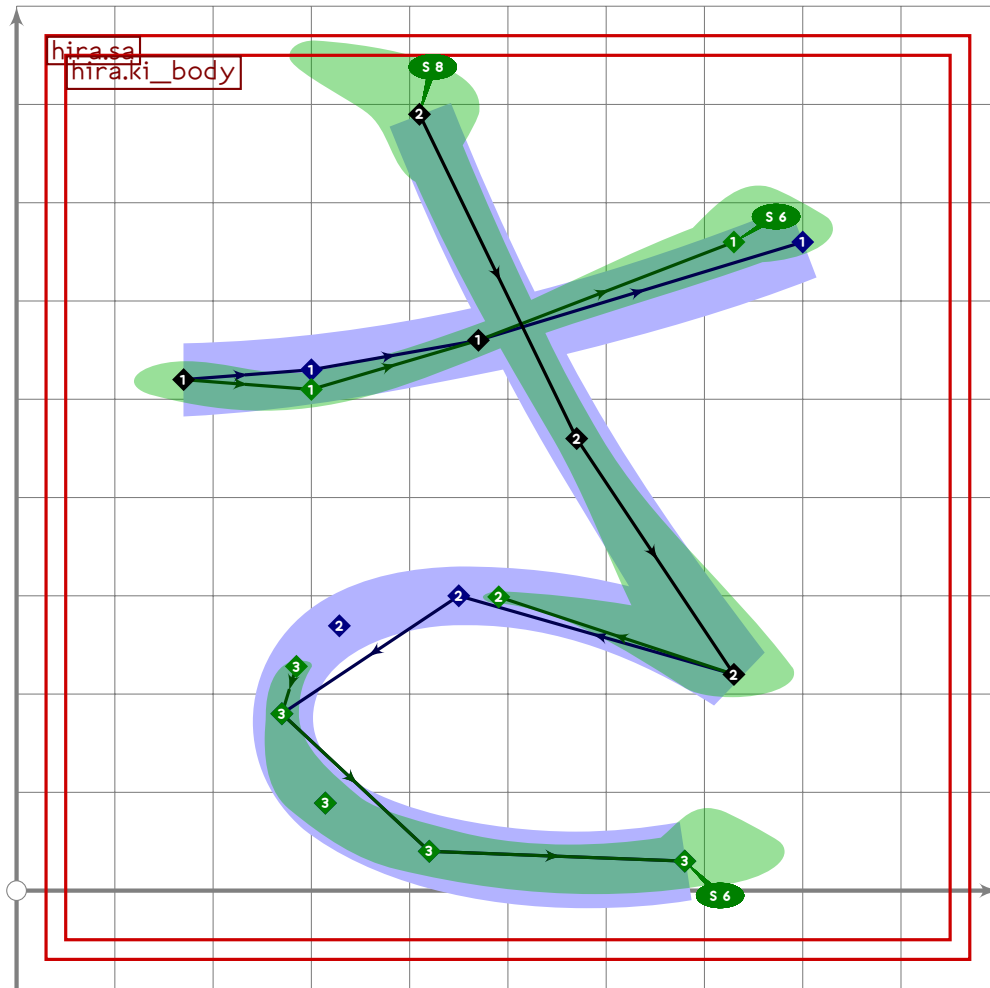
208
209 vardef hira.ko =
210   push_pbox_toexpand("hira.ko");
211
212   push_stroke((280,650)..(460,640)..{curl 1}(720,670){curl 1}..
213     (450,500)..(330,70)..tension 24..(820,60),
214     (1.8,1.8)-(1.9,1.9)-(2.3,2.3)-
215     (0.35,0.25)-(2.2,1.8)..(2.8,2.4));
216   set_botip(0,2,0);
217   set_boserif(0,5,6);
218   expand_pbox;
219 enddef;
220

```

HIRA

Hiragana Sashisuseso/Zajizuzezo

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

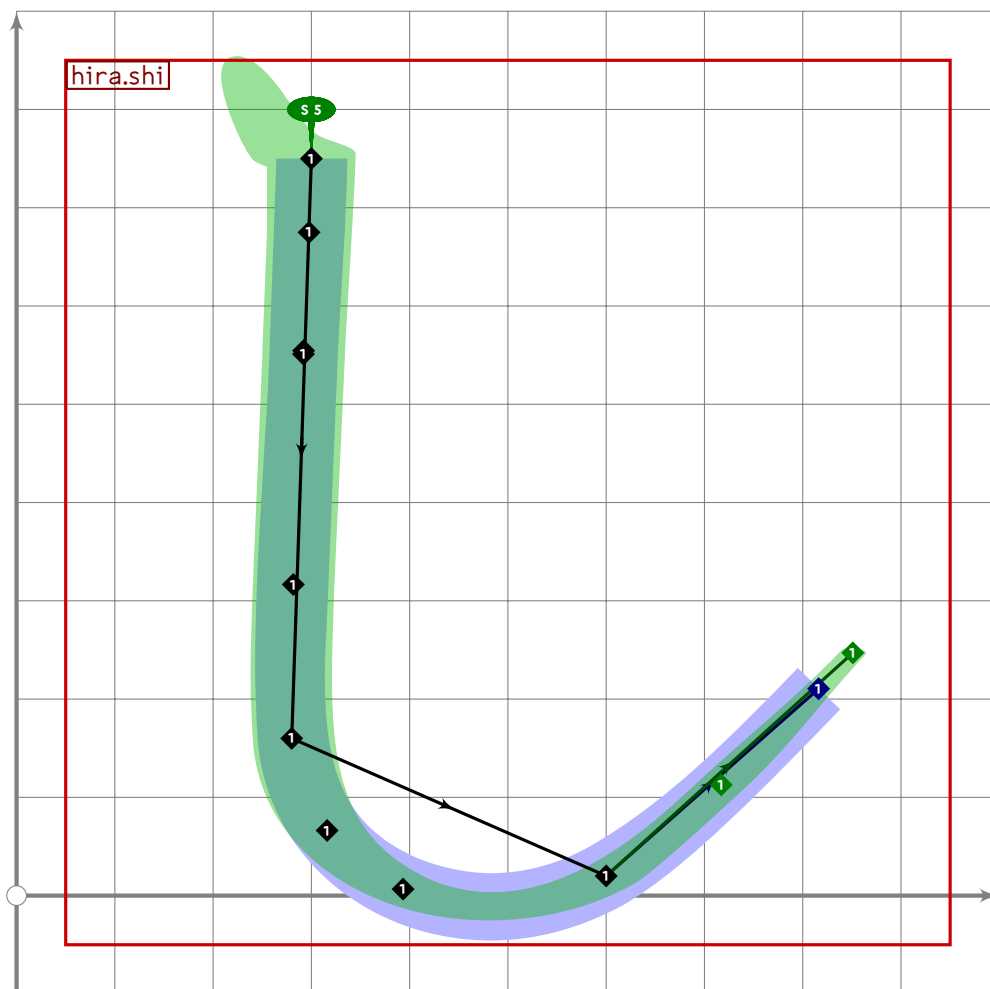


```

222
223 vardef hira.sa =
224   push_pbox_toexpand("hira.sa");
225
226   push_stroke((170,520)..(300,530-20*mincho)..(470,560)..(800-70*mincho,660),
227     (1.9,1.9)-(1.8,1.8)-(1.4,1.4)-(2.1,2.1));
228   set_boserif(0,3,6);
229
230   hira.ki_body;
231   expand_pbox;
232 enddef;

```

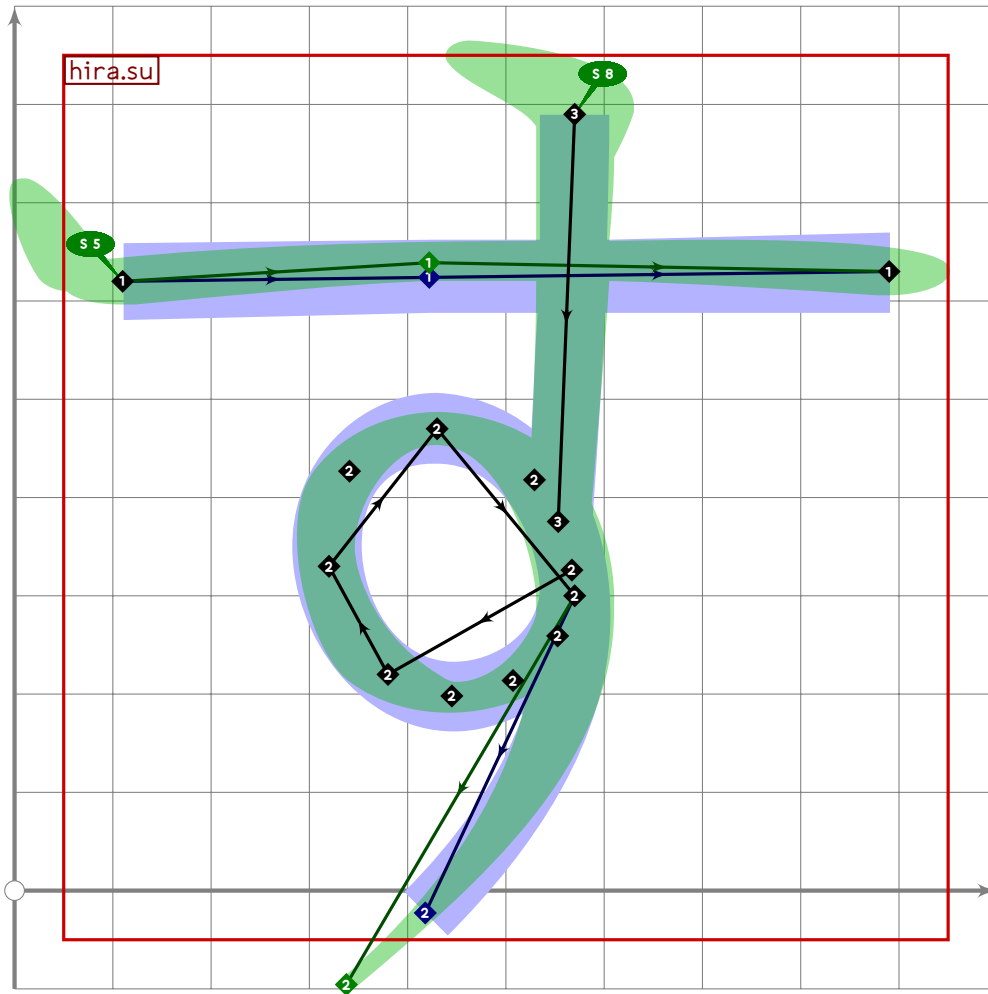
HIRA



```

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

```

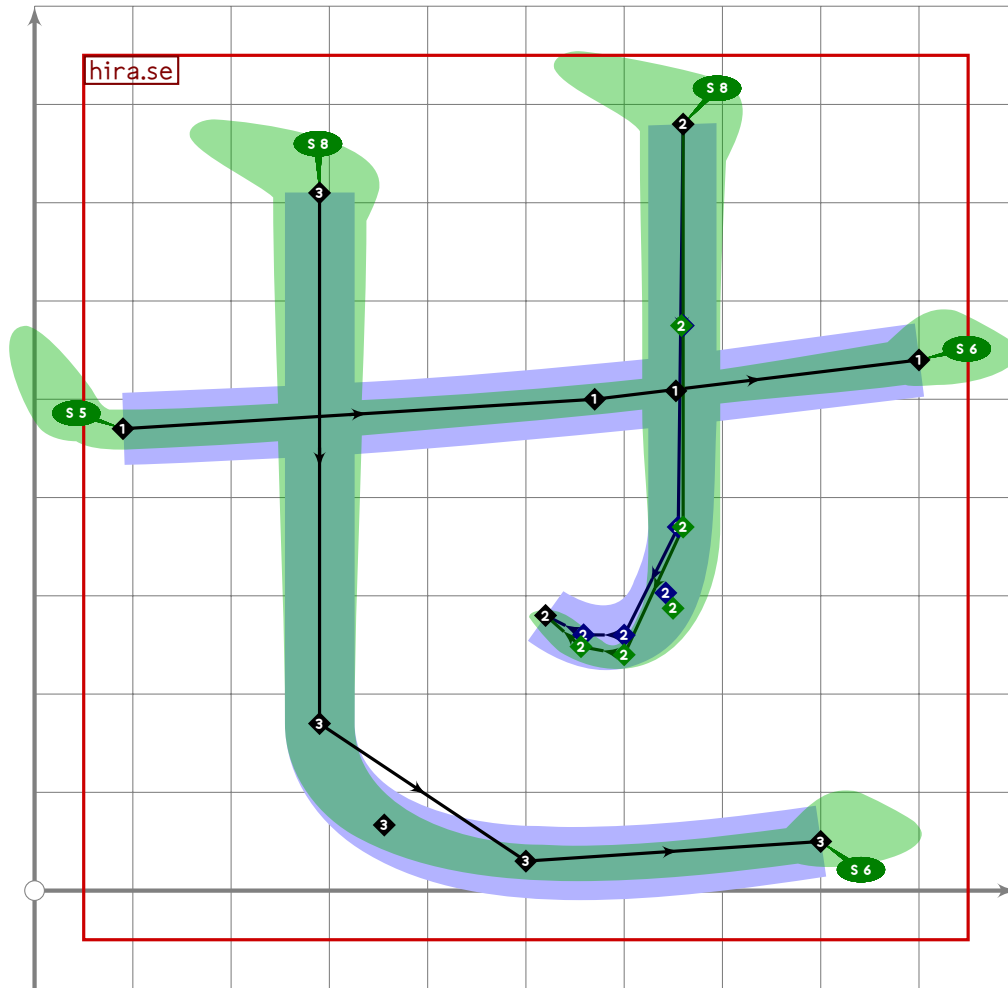


```

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

```

HIRA

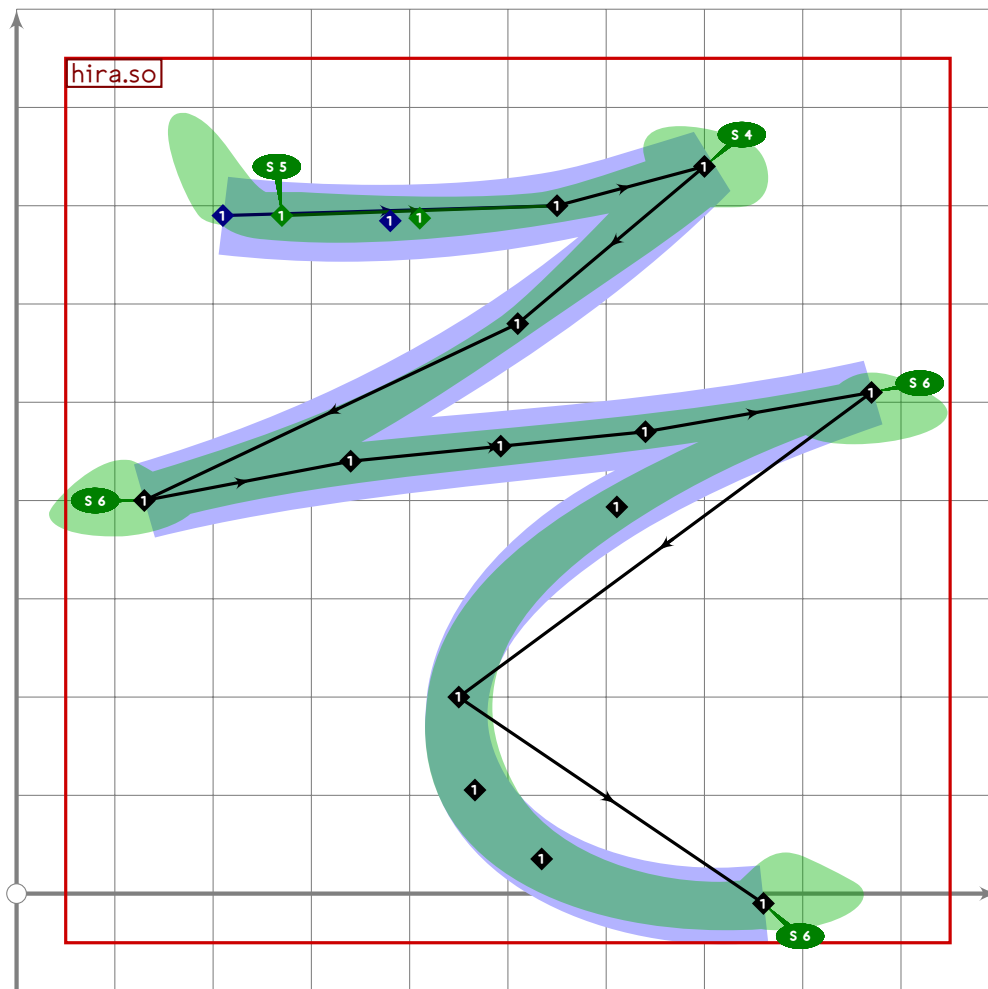


```

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

```

HIRA



```

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

```

HIRA

```

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

```

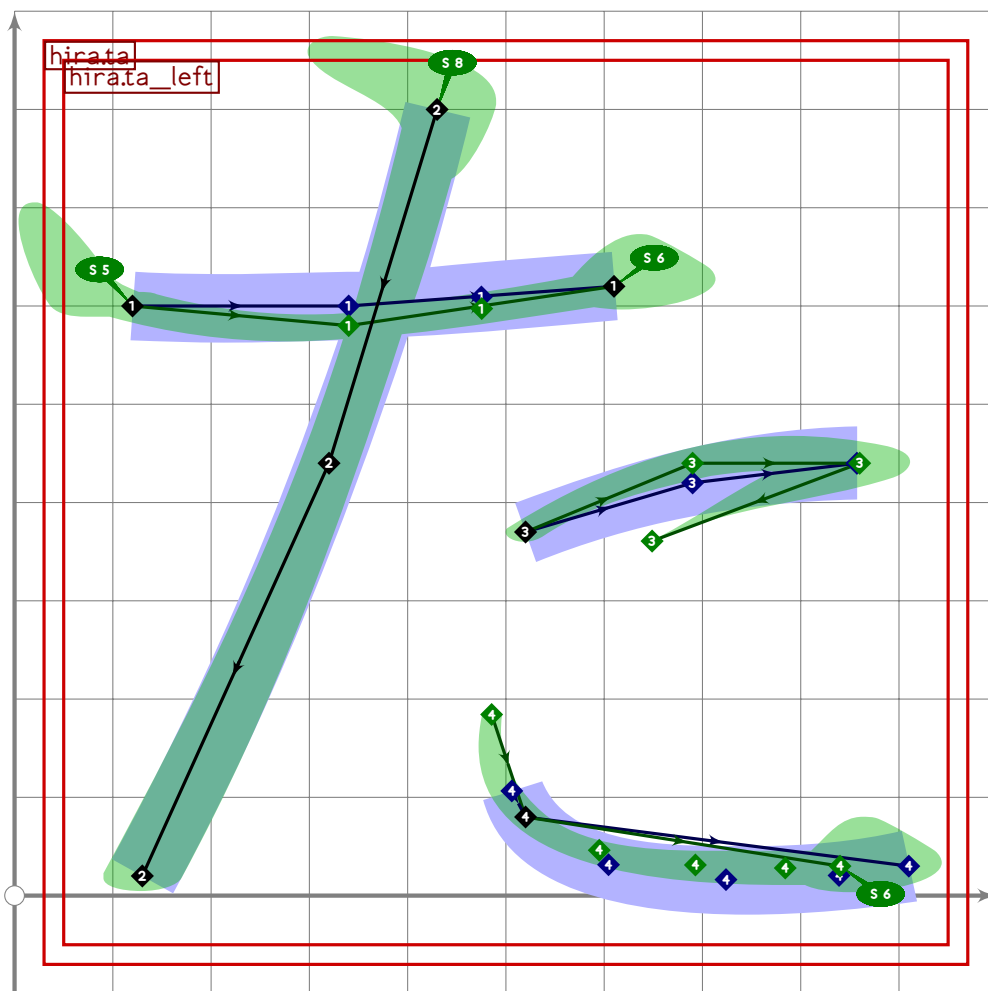
Hiragana Tachitsuteto/Dajizudedo

HIRA

```

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

```

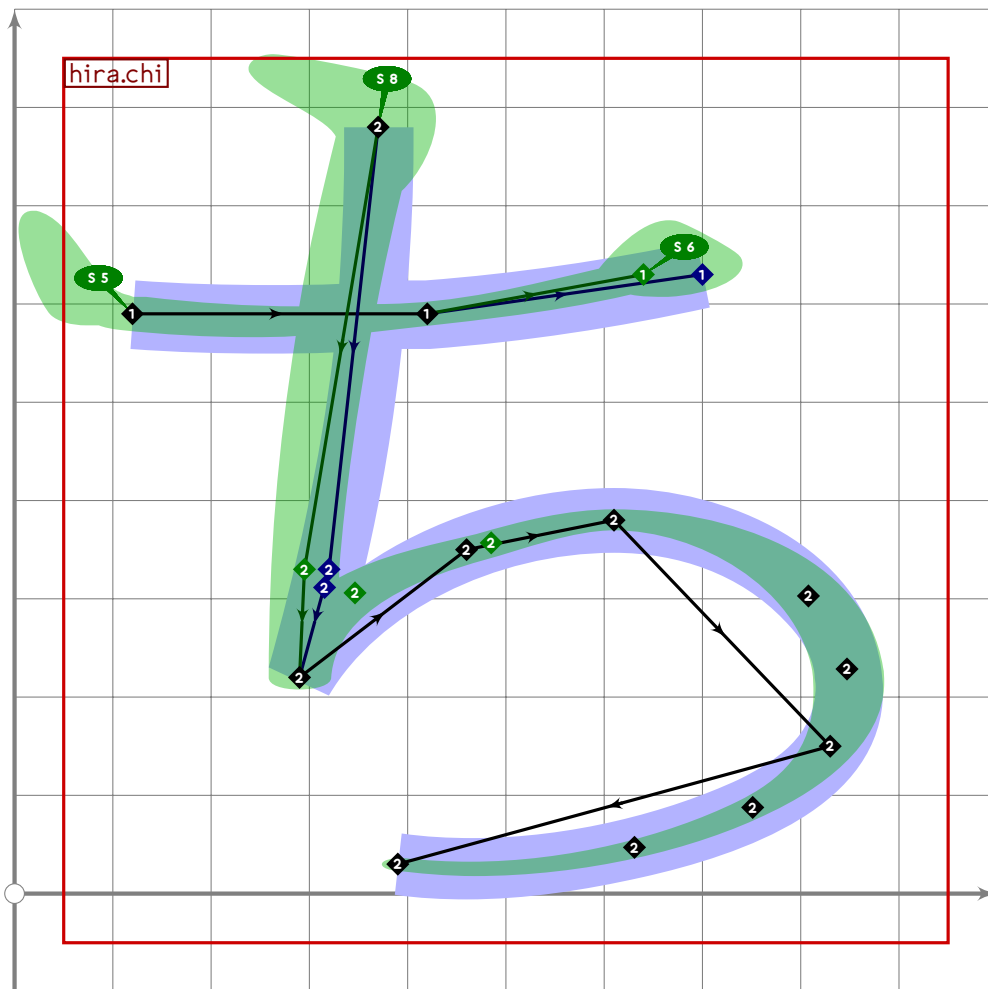


```

321
322 vardef hirata =
323   push_pbox_toexpand("hirata");
324
325   hirata_left;
326
327   push_stroke((520,370)..(690,420+20*mincho)..{curl 1.5}(860,440){curl 0}..
328     (610,280+60*mincho)..(520,80)..tension 1.2 and 3..
329     {curl 0.2}(910-70*mincho,30),
330     (1,1,1)-(1.6,1.6)-(2.8,0.99)-(0.45,0.35)-(1,1,1)-(1.9,1.9));
331   set_botip(0,2,0);
332   set_boserif(0,5,6);
333   expand_pbox;
334 enddef;
335
336 vardef hira.chi_bottom =
337   replace_strokep(0)((oldp){-direction infinity of oldp xscaled 2}..
338     (460,350)..(610,380){right}..(830,150)..
339     tension 14..{curl 0.3}(390,30));
340   replace_strokeq(0)((oldq)..(1.3,1.3)..(1.5,1.5)..(1.5,1.5)..(1,1));
341 enddef;

```

HIRA

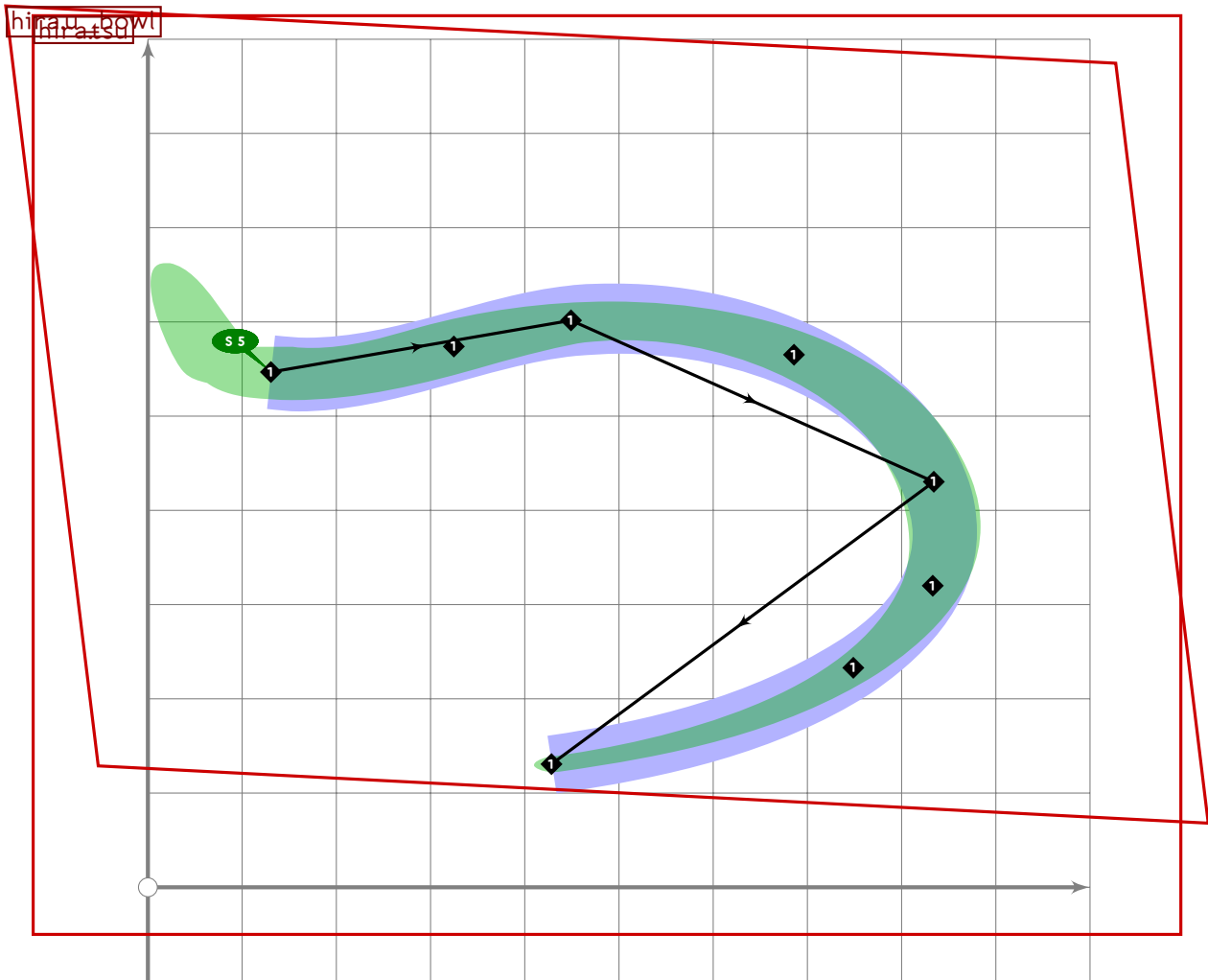


```

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

```

HIRA

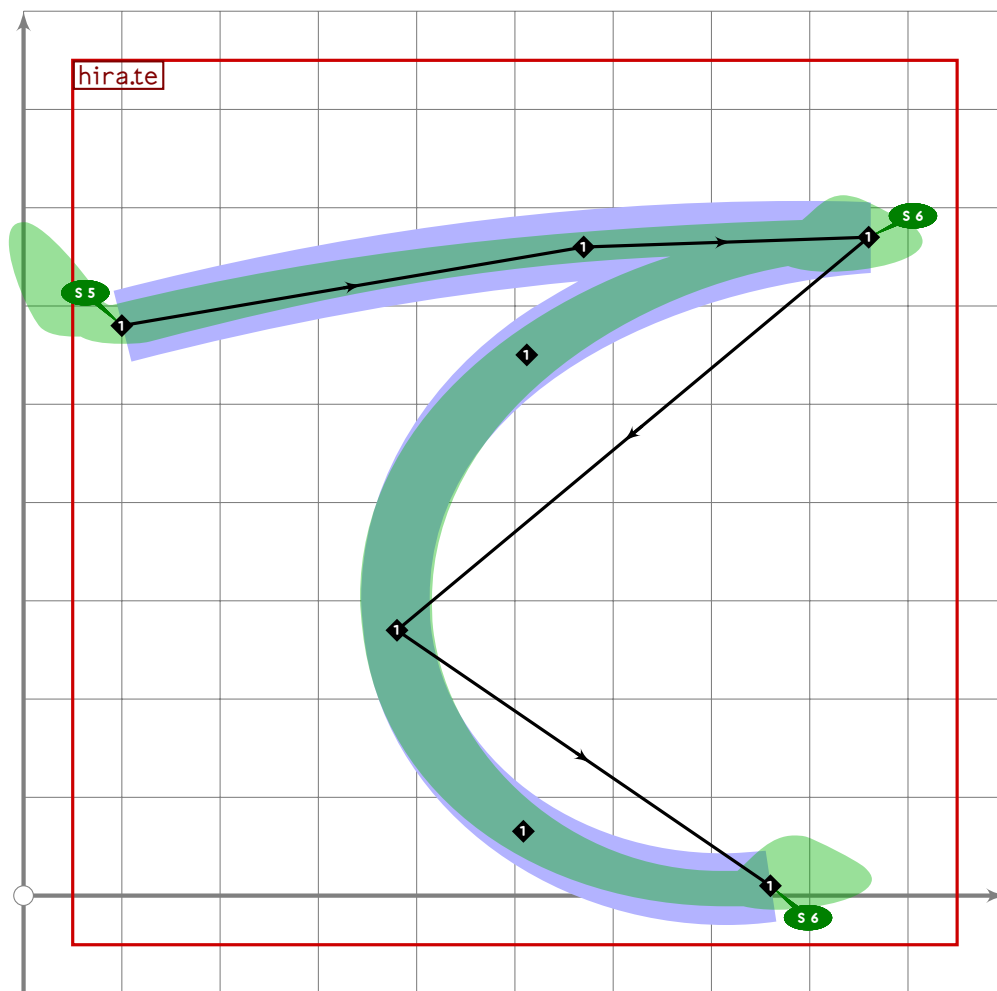


```

358
359 vardef hira.tsu =
360   push_pbox_toexpand("hira.tsu");
361
362   begingroup
363     save xf;
364     transform xf;
365     (300,450) transformed xf=(220,560);
366     (750,350) transformed xf=(820,440);
367     (400,0) transformed xf=(400,150);
368     tsu_xform(xf)(hira.u_bowl);
369     set_bosize(0)(100+10*mincho);
370   endgroup;
371   expand_pbox;
372 enddef;

```

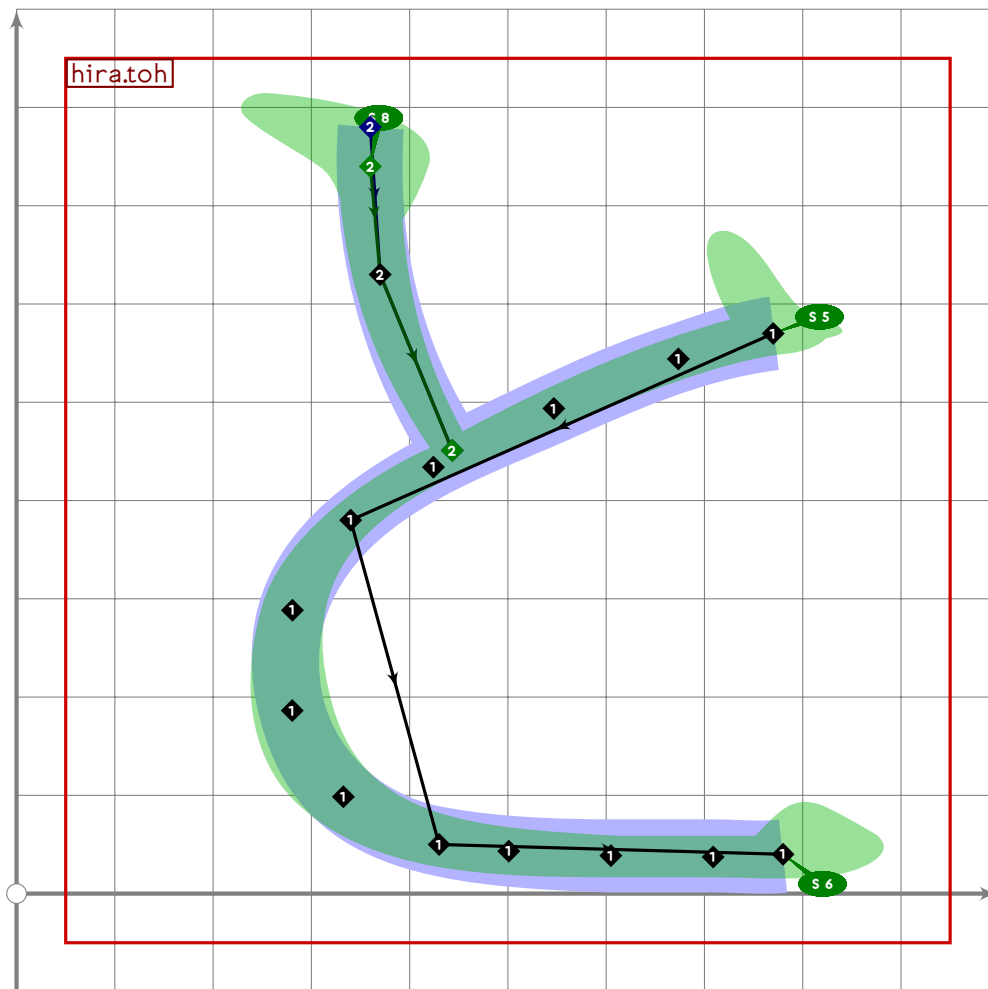
HIRA



```

373
374 vardef hirate =
375   push_pbox_toexpand("hira.te");
376
377   push_stroke((100,580)..(570,660)..{curl 1}{(860,670){curl 0.2}..
378     (380,270)..{curl 0.6}{(760,10),
379     (1,1,9)-(1.5,1.5)-(1.8,1.8)-(1.5,1.5)-(1.8,1.8));
380   set_botip(0,2,0);
381   set_boserif(0,0,5);
382   set_boserif(0,2,6);
383   set_boserif(0,4,6);
384   expand_pbox;
385 enddef;

```



```

386
387 vardef hira.toh =
388   push_pbox_toexpand("hira.toh");
389
390   push_stroke((770,570)..tension 1.7..(340,380)..(430,50)..
391     tension 2..(780,40),
392     (2,2)-(1.2,1.2)-(2.1,2.1)-(2,2));
393   set_boserif(0,0,5);
394   set_boserif(0,3,6);
395
396   push_stroke(subpath (0,197) of
397     ((360,780-40*mincho)..(370,630)..(point 0.7 of get_stroke(0))),
398     (1.4,1.4)-(1.3,1.3)-(1.1,1.1));
399   set_boserif(0,0,8);
400   expand_pbox;
401 enddef;
402

```

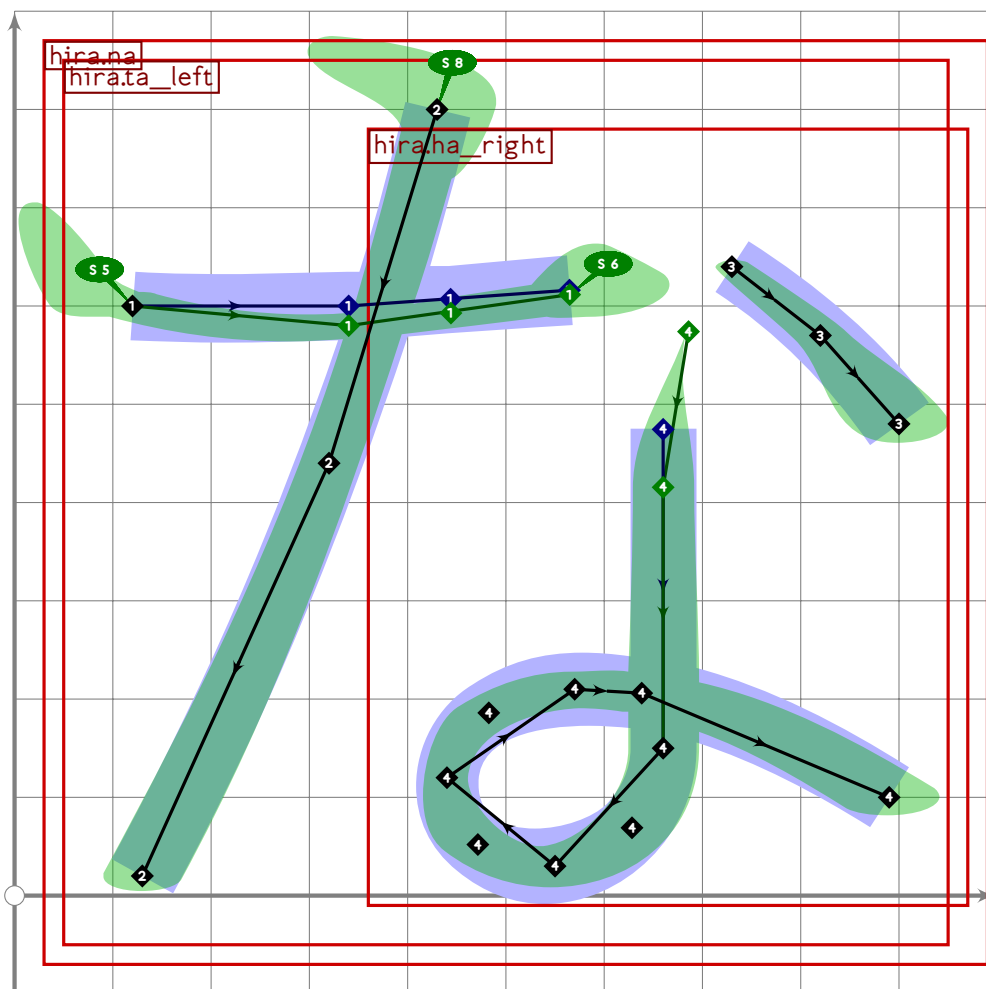
HIRA

Hiragana Naninuneno

```

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

```



```

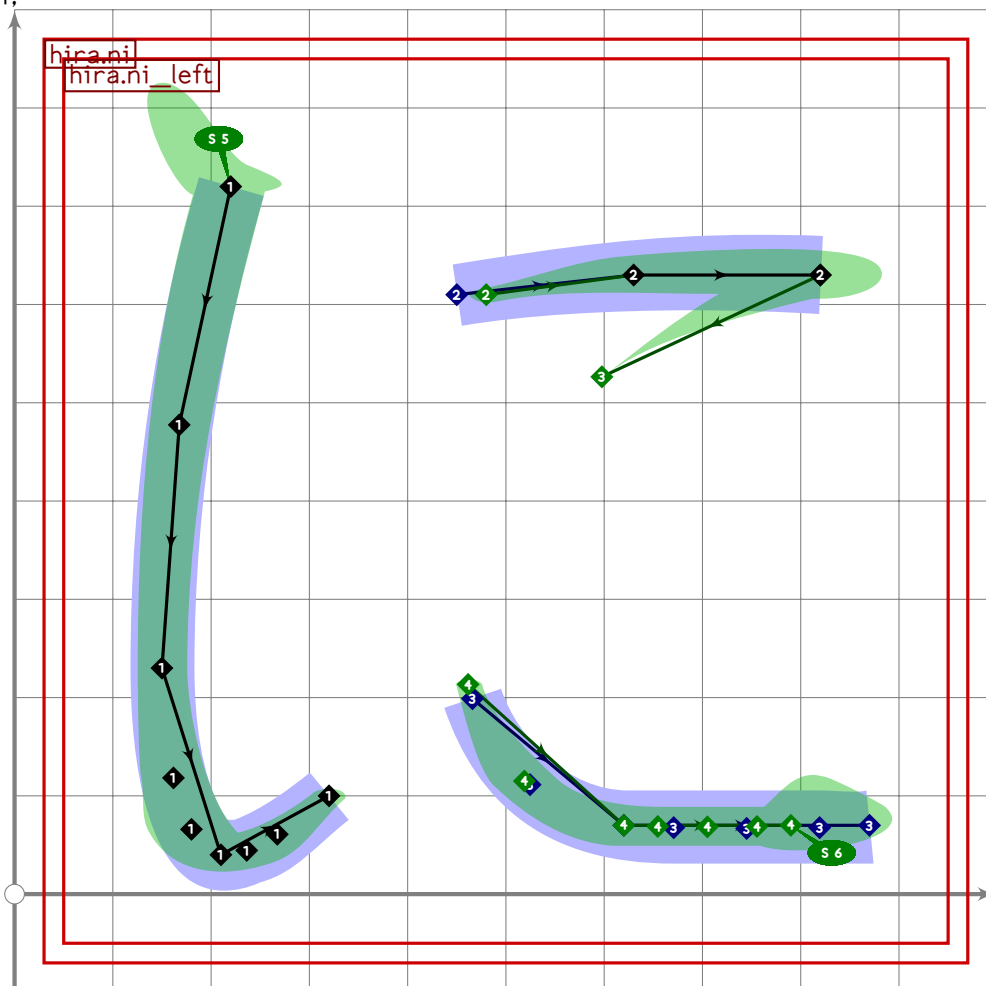
404
405 vardef hira.na =
406   push_pbox_toexpand("hira.na");
407
408   hira.ta_left;
409   replace_stroke(-1)(subpath (0,1,8) of oldp);
410
411   push_stroke((730,640)..(820,570)..(900,480),
412     (1,1)..(1.3,1.3)..(1.9,1.9));
413
414   hira.ha_right;
415
416   replace_stroke(0)((120,0)+point 0 of oldp)
417     ..(subpath (0.45+0.1*mincho,infinity) of oldp));
418   replace_stroke(0)((-0.2,-0.4)-(1.7,1)-(1.5,1.5)-(2,2)-
419     (1.1,1.1)-(1.9,1.9));
420   expand_pbox;
421 enddef;
422
423 vardef hira.ni_left =
424   push_pbox_toexpand("hira.ni_left");

```

```

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

```



```

434
435 vardef hira.ni =
436   push_pbox_toexpand("hira.ni");
437
438   hira.ni_left;
439
440   push_stroke((450+30*mincho,610)..(630,630)..(820,630),
441     (1,1)..(1.9,1.9)..(2.2,2.2));
442
443   push_stroke((point infinity of get_strokep(0)){curl 0.6}..(530,460)..
444     (460,220)..(620,70)..tension 2..(870-80*mincho,70),
445     (3,0.7)-(0,0)-(0.8,0.9)-(2,2)-(1.9,1.9));

```

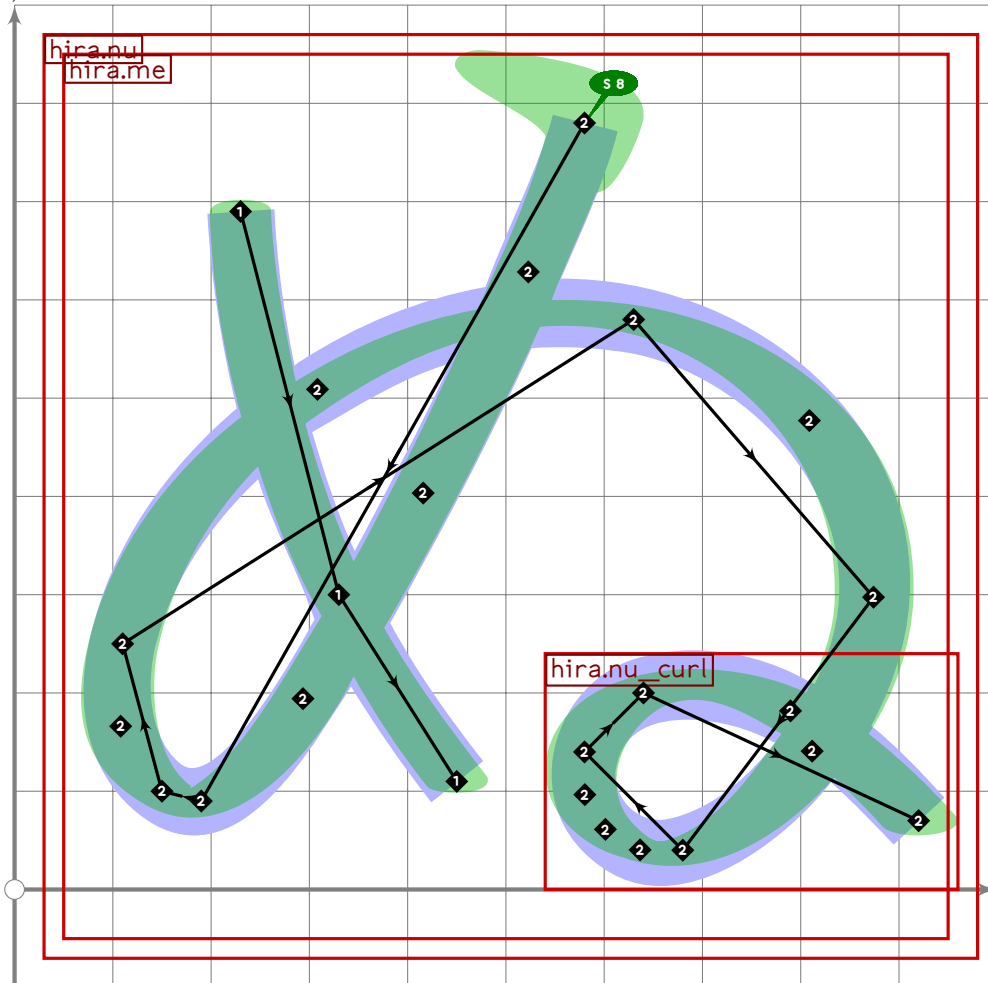
HIRA

U+306C
tsuku.uni306C

```

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

```



HIRA

```

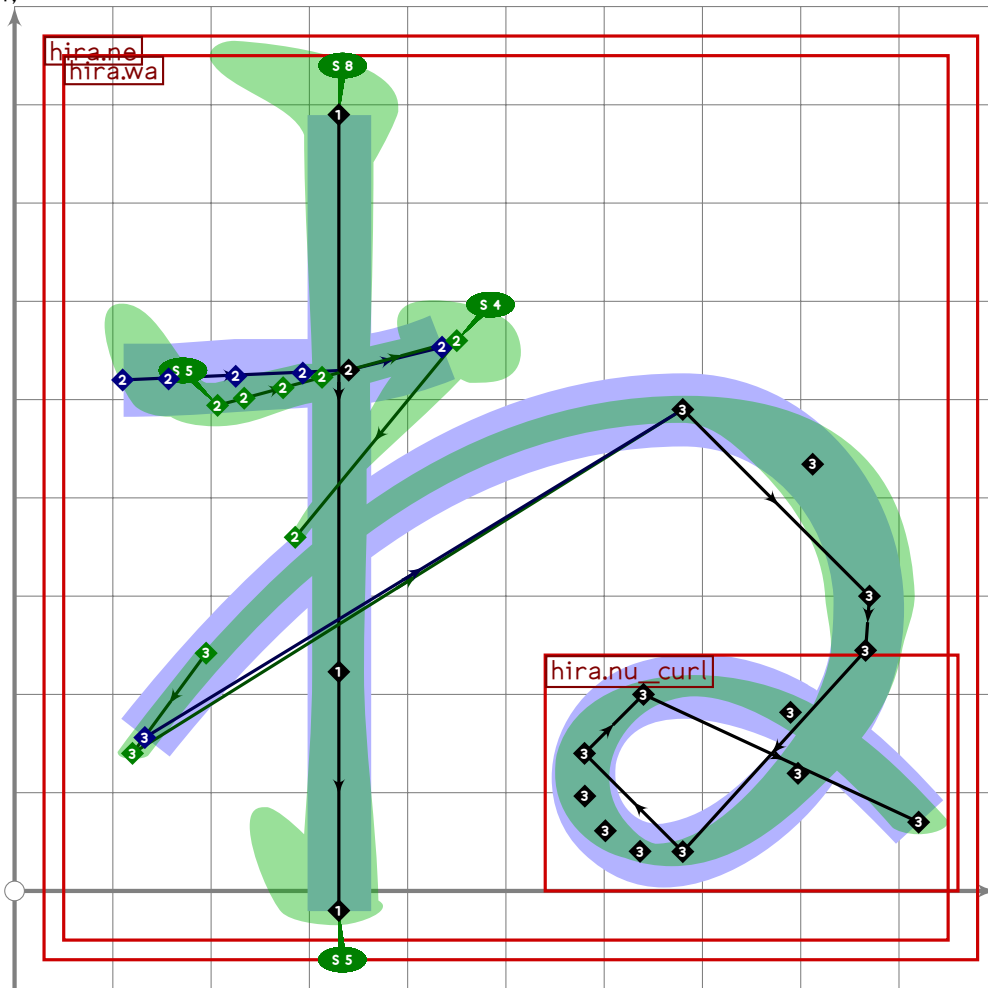
457
458 vardef hira.nu =
459   push_pbox_toexpand("hira.nu");
460
461   hira.me;
462
463   replace_strokeq(0)((subpath (0,4,8) of oldp)..tension 1.2..
464     hira.nu_curl);
465   replace_strokeq(0)((1.5,1.5)-(1.4,1.4)-(1.6,1.6)-(1.4,1.4)-
466     (1.6,1.6)-(1.6,1.6)-(1.7,1.7)-(1.3,1.3)-(1.6,1.6));

```

```

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

```

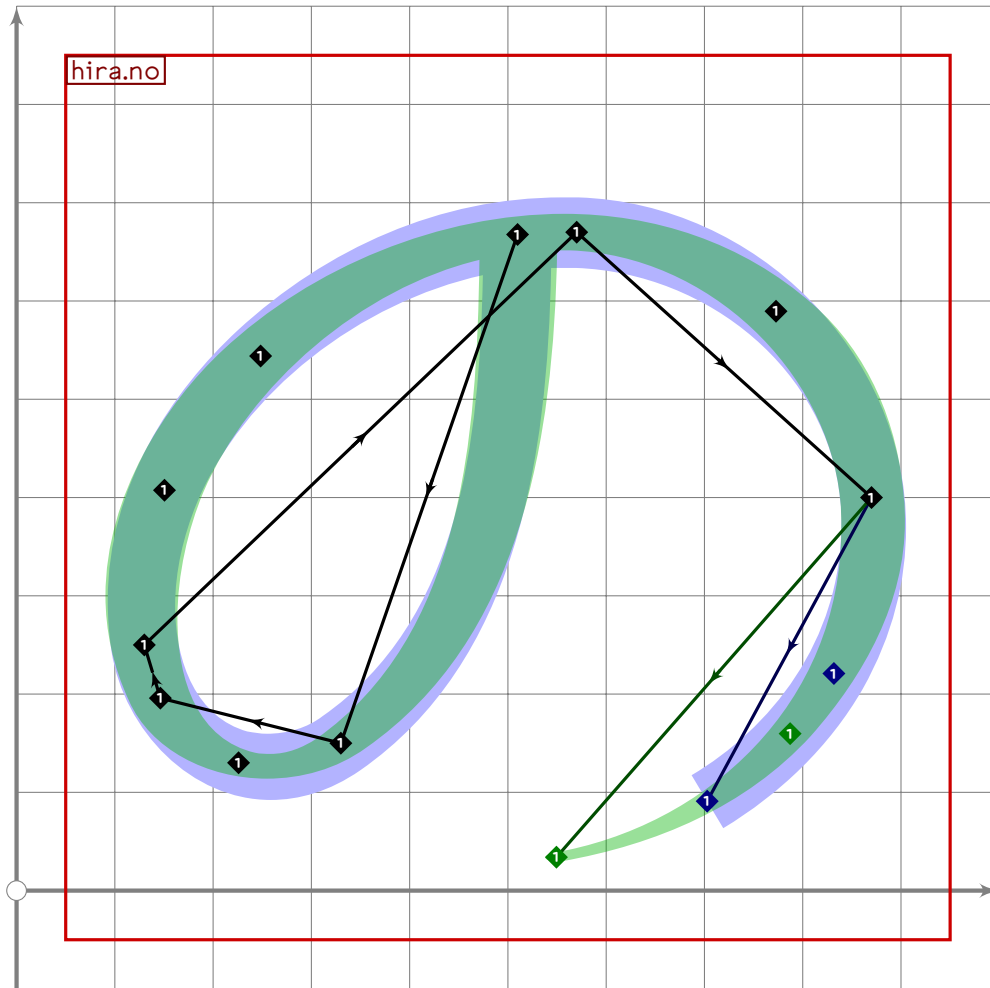


```

470
471 vardef hira.ne =
472   push_pbox_toexpand("hira.ne");
473
474   hira.wa;
475
476   replace_strokeq(0)((subpath (0,6,1) of oldp)..tension 1.2..
477     hira.nu_curl);
478   replace_strokeq(0)((2,2)-(1.6,1.6)-(2.2,0.9)-(0.7,0.7)-(0.97,0.97)-
479     (2,2)-(1.5,1.5)-(1.4,1.4)-(1.4,1.4)-(1.2,1.2)-(1.3,1.3));
480   expand_pbox;
481 enddef;

```

HIRA



```

482
483 vardef hira.no =
484   push_pbox_toexpand("hira.no");
485
486   begingroup
487     save px,py;
488     path px,py;
489     px:=(410,30)..(130,250)..tension 11..(570,670)..(870,400)..cycle;
490     py:=(510,770){down}{dir 215}(330,150);
491
492     px:=subpath (0.854) of px;
493     push_stroke(
494       (subpath (xpart (py intersectiontimes px),infinity) of py)..px,
495       (1.6,1.6)-(1.3,1.3)-(1.4,1.4)-(1.5,1.5)-(1.8,1.8)-
496       (1.4,1.4)-(0.7,0.7));
497   endgroup;
498   expand_pbox;
499 enddef;
500

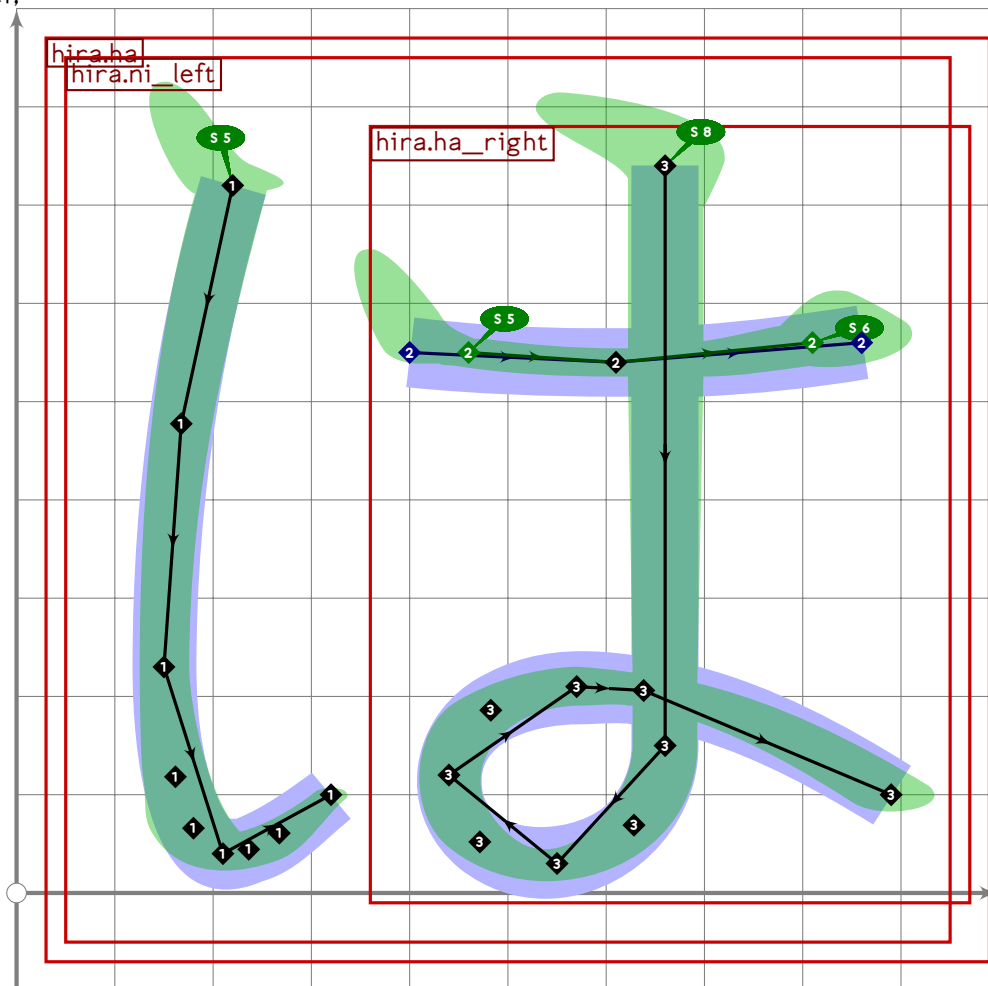
```

Hiragana Hahifuheho/Babibubebo/Papipupepo

```

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

```



```

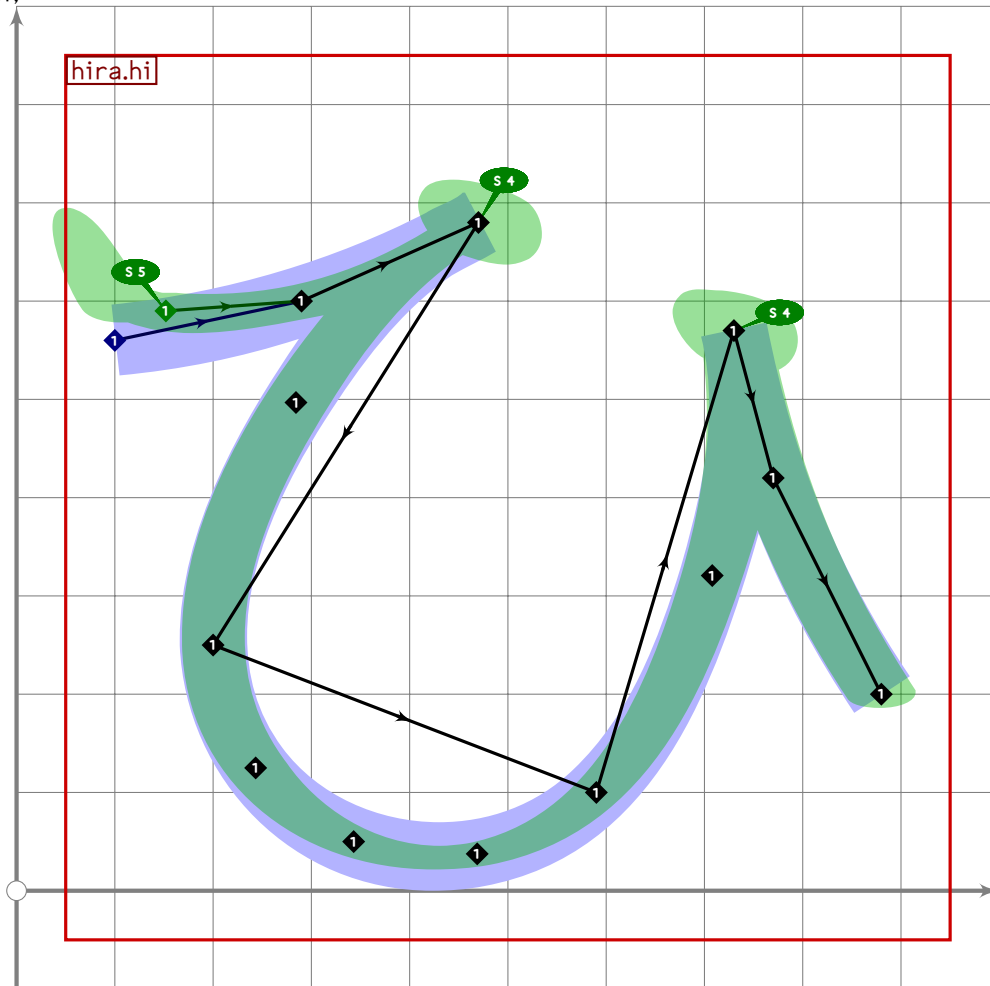
512
513 vardef hira.ha =
514   push_pbox_toexpand("hira.ha");
515
516   hira.ni_left;
517
518   push_stroke((400+60*mincho,550)..(610,540)..(860-50*mincho,560),
519     (1.6,1.6)..(1.6,1.6)..(2,2));

```

HIRA

U+3072
tsuku.uni3072

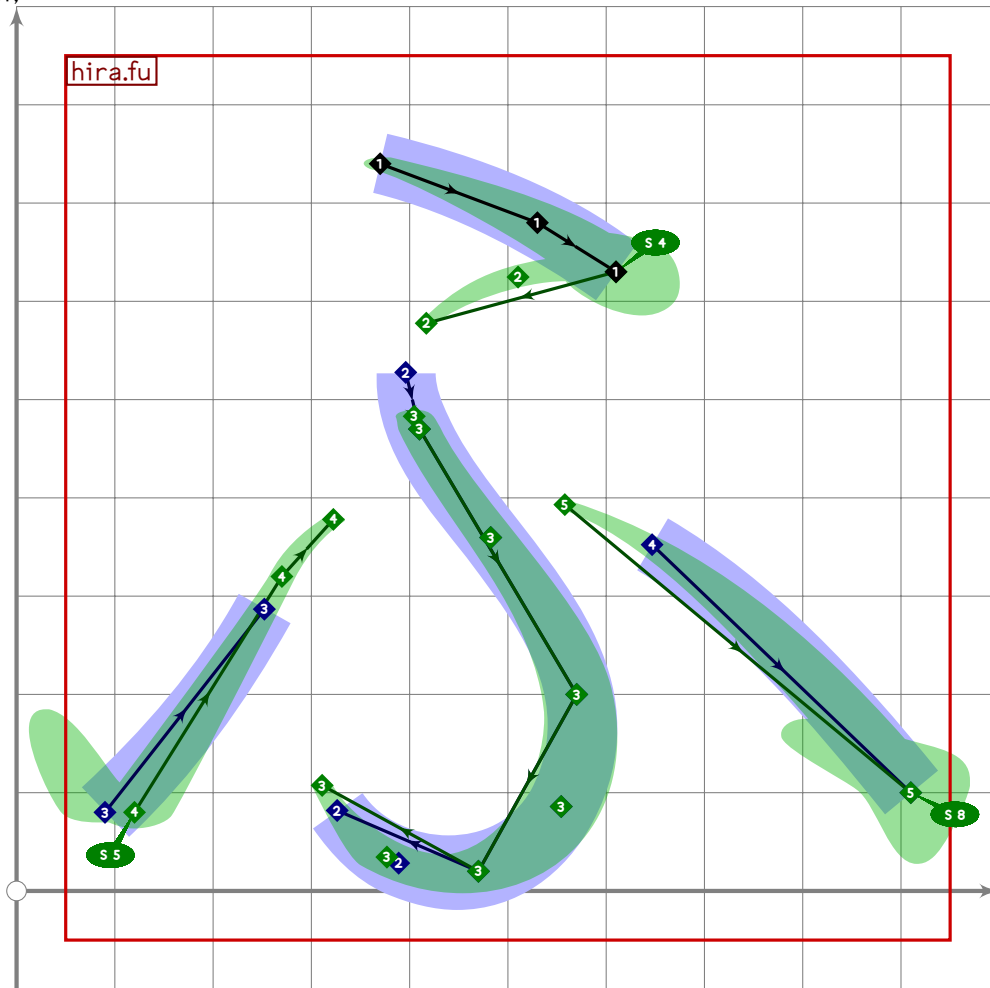
```
520 set_boserif(0,0,5);
521 set_boserif(0,2,6);
522
523 hira.ha_right;
524 set_boserif(0,0,8);
525 expand_pbox;
526 enddef;
```



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

HIRA

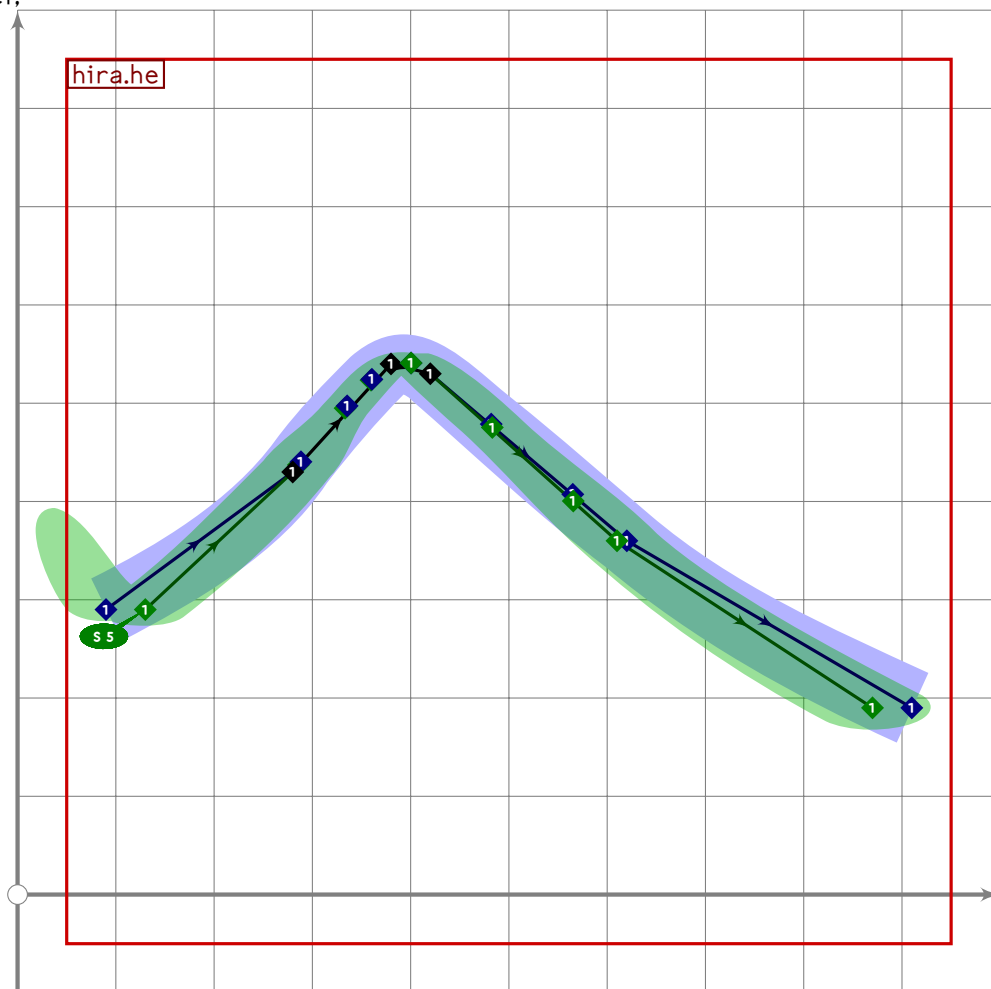
```
541 set_boserif(0,5,4);
542 expand_pbox;
543 enddef;
```



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

HIRA

562 enddef;



563

564 vardef hira.he =

565 push_pbox_toexpand("hira.he");

566

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

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

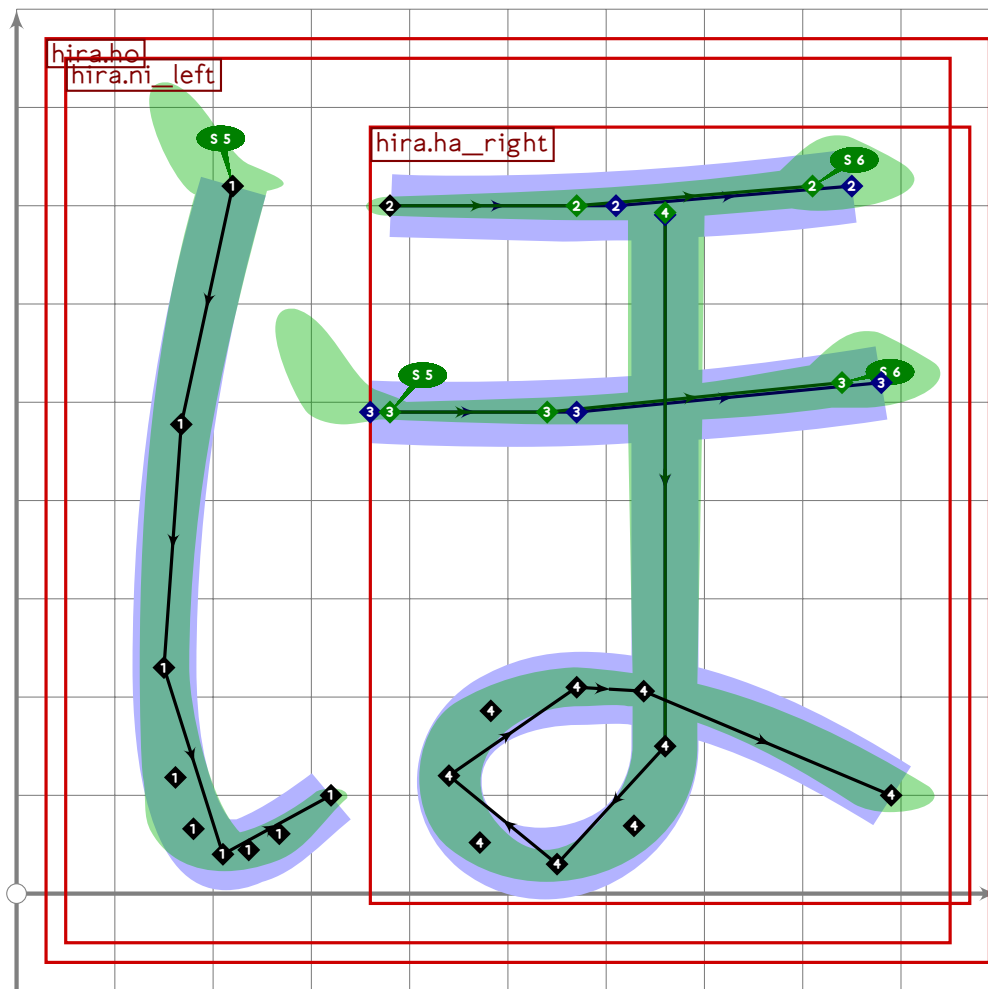
569 (1.7,1.7)..(1.6,1.6)..(1.2,1.2)..(1.3,1.3)..(1.7,1.7)..(2.1,2.1));

570 set_boserif(0,0,5);

571 expand_pbox;

572 enddef;

HIRA



```

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

```

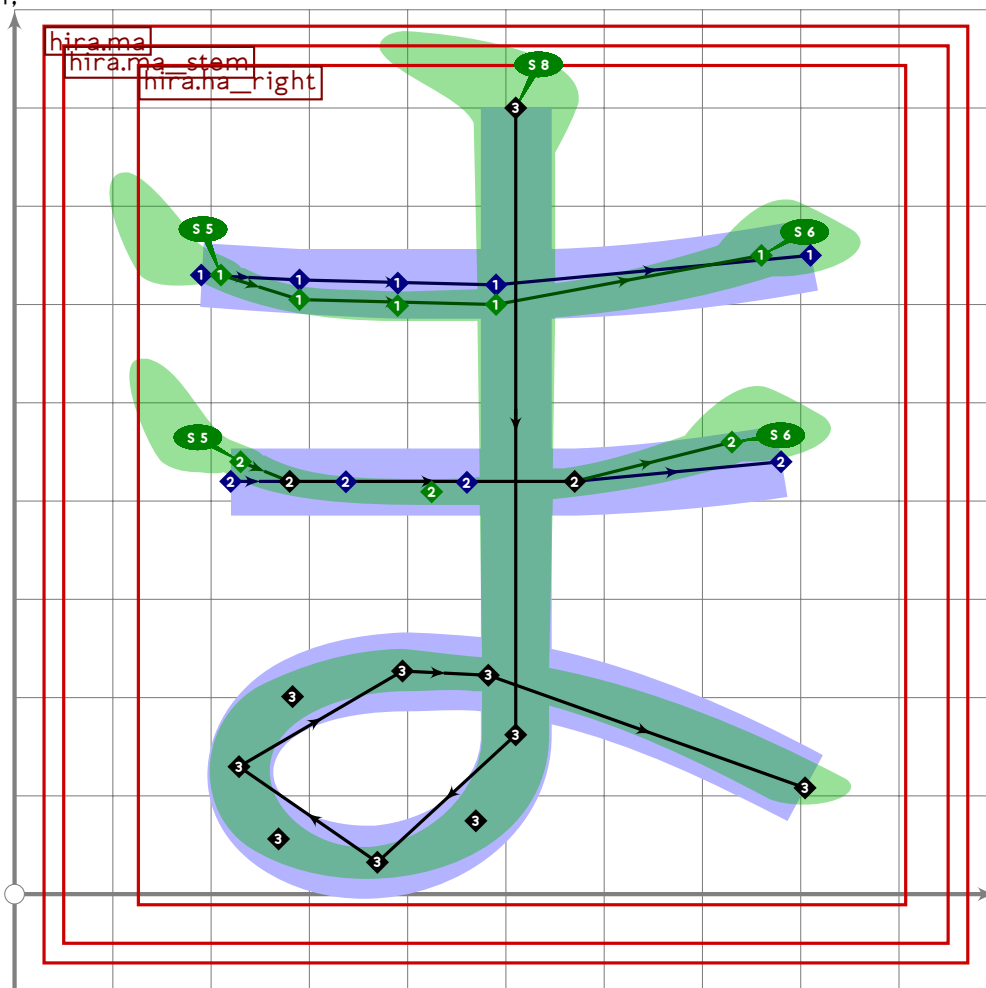
HIRA

Hiragana Mamimumemo

```

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

```



```

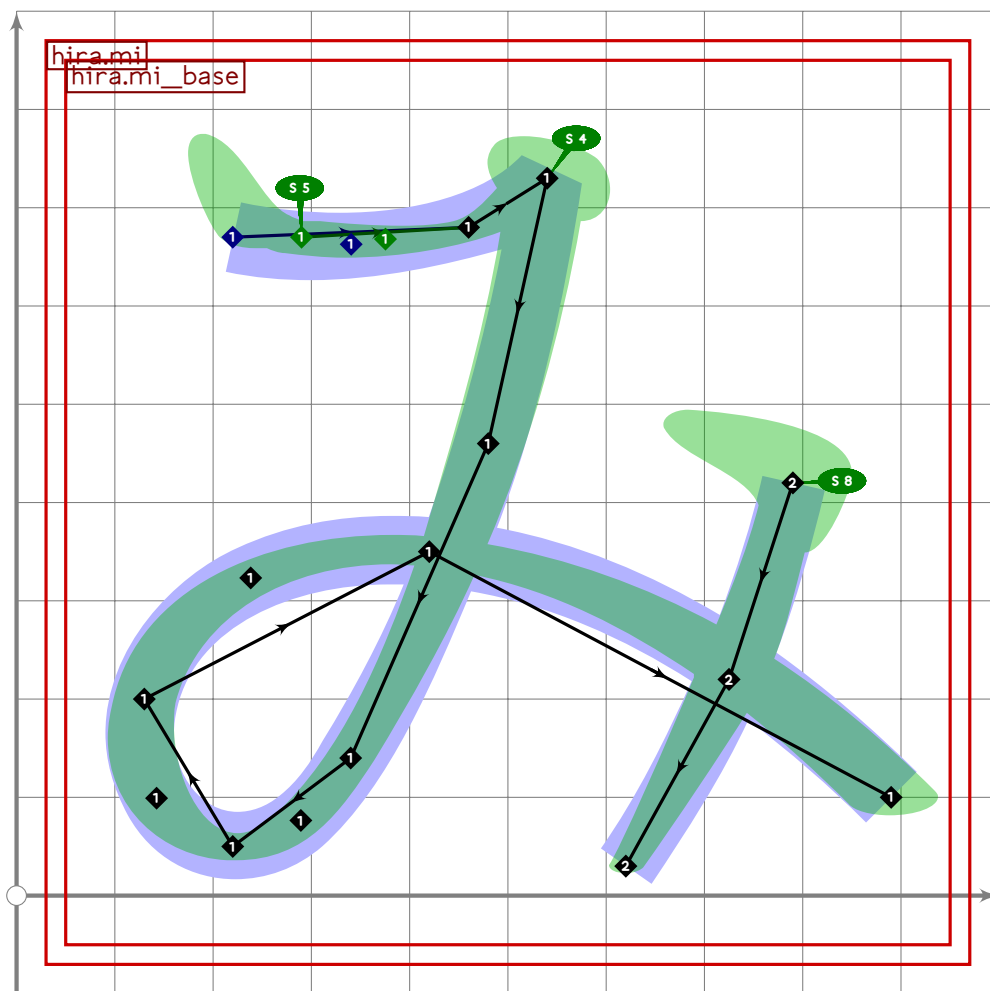
611
612 vardef hira.ma =

```

```

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

```

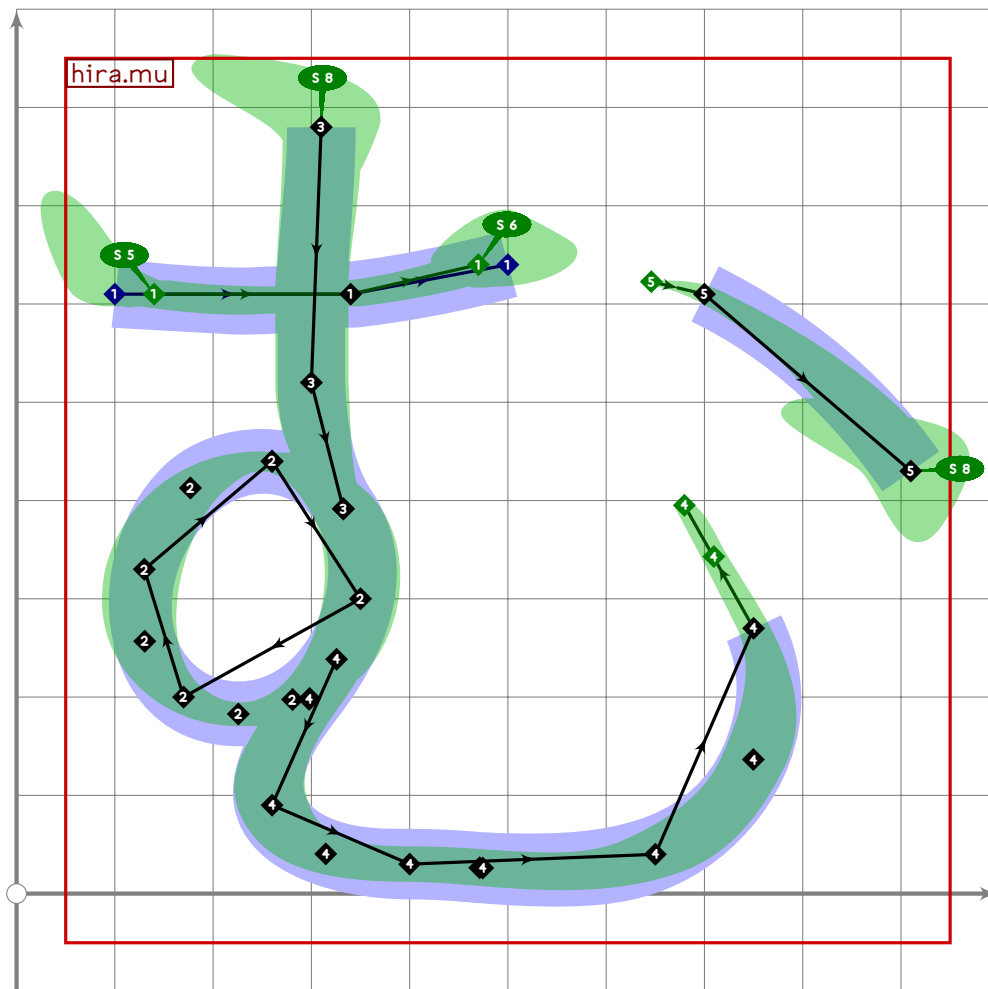



```

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

```

HIRA



```

658
659 vardef hira.mu =
660   push_pbox_toexpand("hira.mu");
661
662   push_stroke((100+40*mincho,610)..(340,610)..(500-30*mincho,640),
663     (1,6,1,6)-(1,4,1,4)-(1,5,1,5));
664   set_boserif(0,0,5);
665   set_boserif(0,2,6);
666
667   push_stroke((260,440)..(350,300)..(170,200)..(130,330)..cycle,
668     (1,3,1,3)..(1,6,1,6)..(1,3,1,3)..(1,6,1,6)..cycle);
669
670   push_stroke((310,780){down}..(300,520)..
671     (point 0.5 of get_stroke(0)){direction 0.5 of get_stroke(0)},
672     (1,6,1,6)-(1,5,1,5)-(1,2,1,2));
673   set_boserif(0,0,8);
674
675   push_stroke(
676     (point 1.25 of get_stroke(-1)){direction 1.20 of get_stroke(-1)}..
677     (260,90)..(400,30){right}..(650,40)..(750,270)..
678     tension 2..(600,590)..(700,610)..(910,430),

```

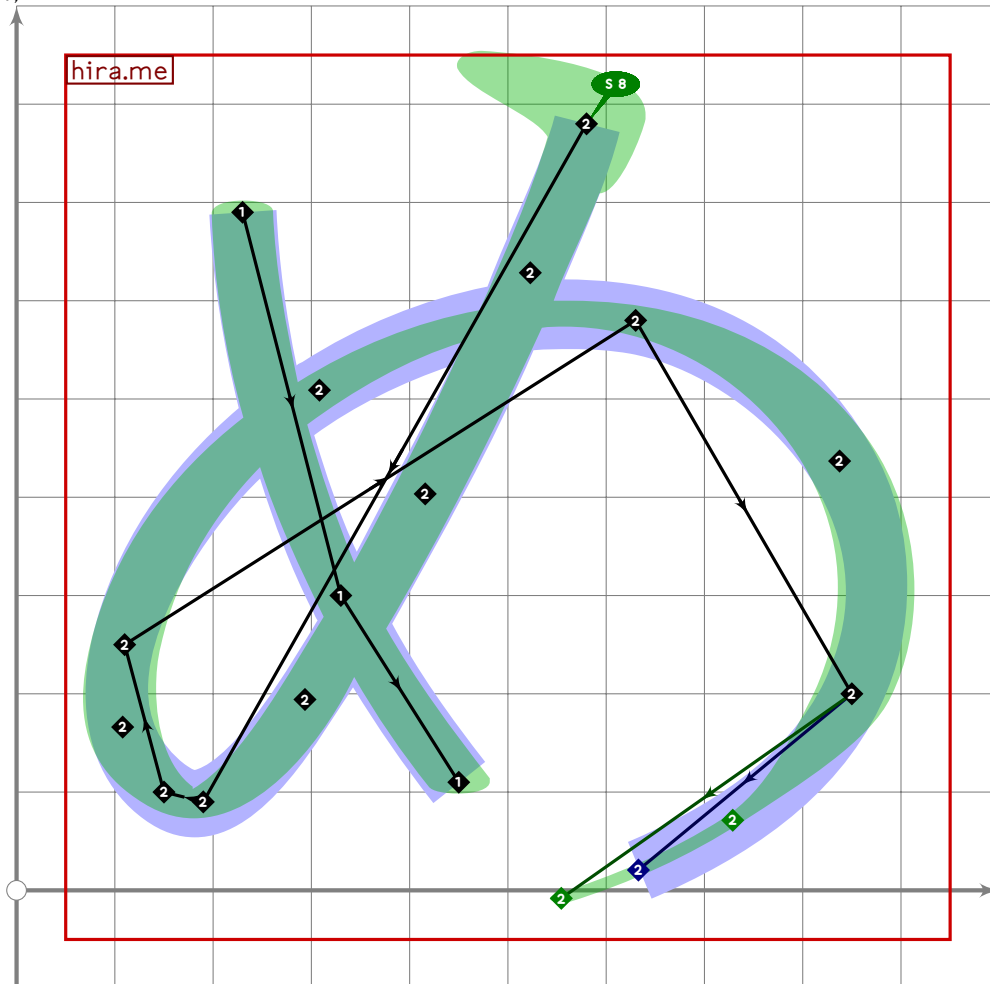
HIRA

U+3081
tsuku.uni3081

```

679 (14,14)-(1.5,1.5)-(1.8,1.8)-(1.9,1.9)-
680 (1,1)-(0.6,0.6)-(1,1)-(1.6,1.6));
681 set_boserif(0,78);
682 expand_pbox;
683 enddef;

```

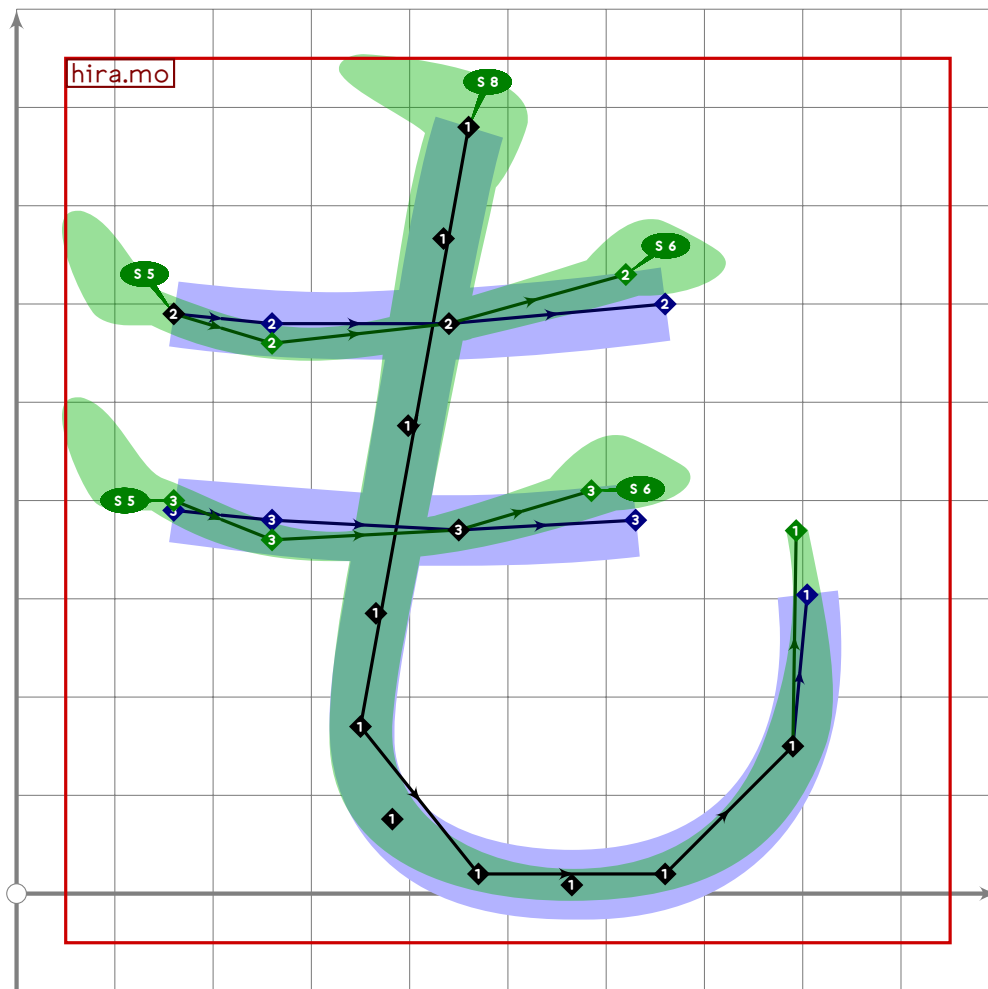


```

684
685 vardef hira.me =
686   push_pbox_toexpand("hira.me");
687
688   push_stroke((230,690)..(330,300)..(450,110),
689     (14,14)-(1.3,1.3)-(14,14));
690
691   push_stroke((580,780){curl 0.2}.tension 2.5..(190,90)..
692     (150,100)..(110,250)..tension 1.1..(630,580)..
693     (850,200).tension 1.1.{curl 0.2}(470,30),
694     (1.5,1.5)-(14,14)-(1.6,1.6)-(14,14)-
695     (1.6,1.6)-(1.6,1.6)-(0.65,0.65));
696   set_boserif(0,0,8);
697   expand_pbox;
698 enddef;

```

HIRA



```

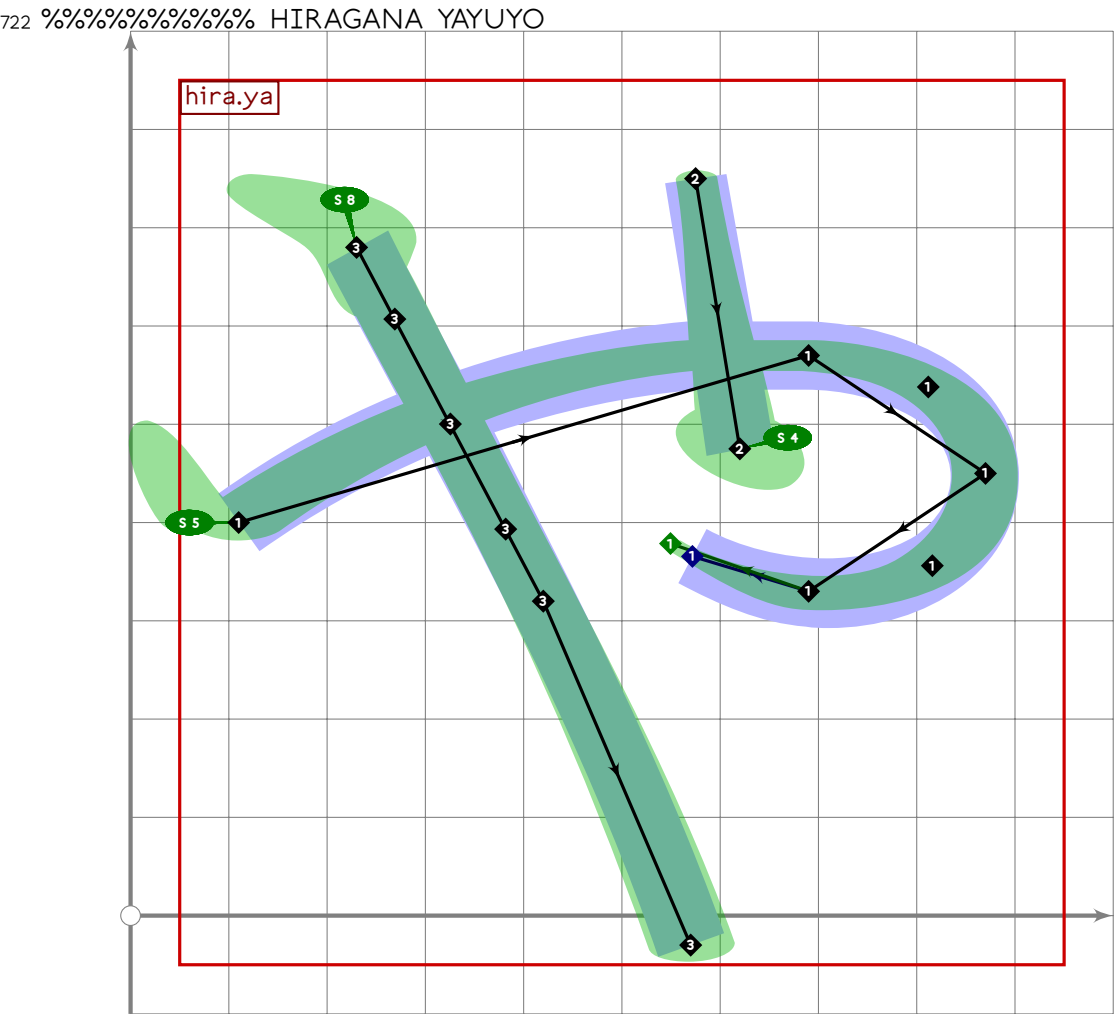
699
700 vardef hira.mo =
701   push_pbox_toexpand("hira.mo");
702
703   push_stroke((460,780)..tension 3..(350,170)..(470,20)..
704     tension 1.2..(660,20)..(790,150)..{curl 0}(770,460),
705     (1,7,1,7)-(1,4,1,4)-(1,6,1,6)-(1,7,1,7)-(1,4,1,4)-(0,6,0,6));
706   set_boserif(0,0,8);
707
708   push_stroke((160,590)..(260,580-20*mincho)..(440,580)..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

```

HIRA

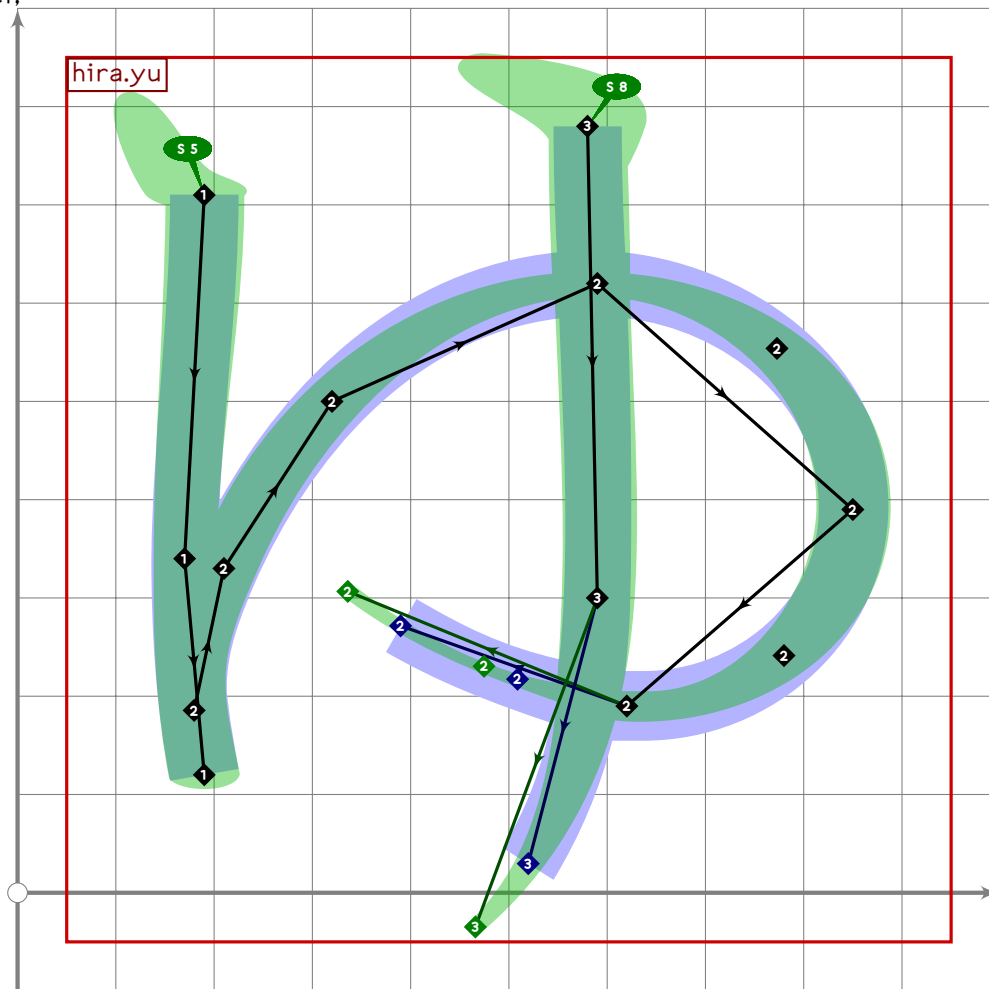
720 enddef;
721

Hiragana Yayuyo



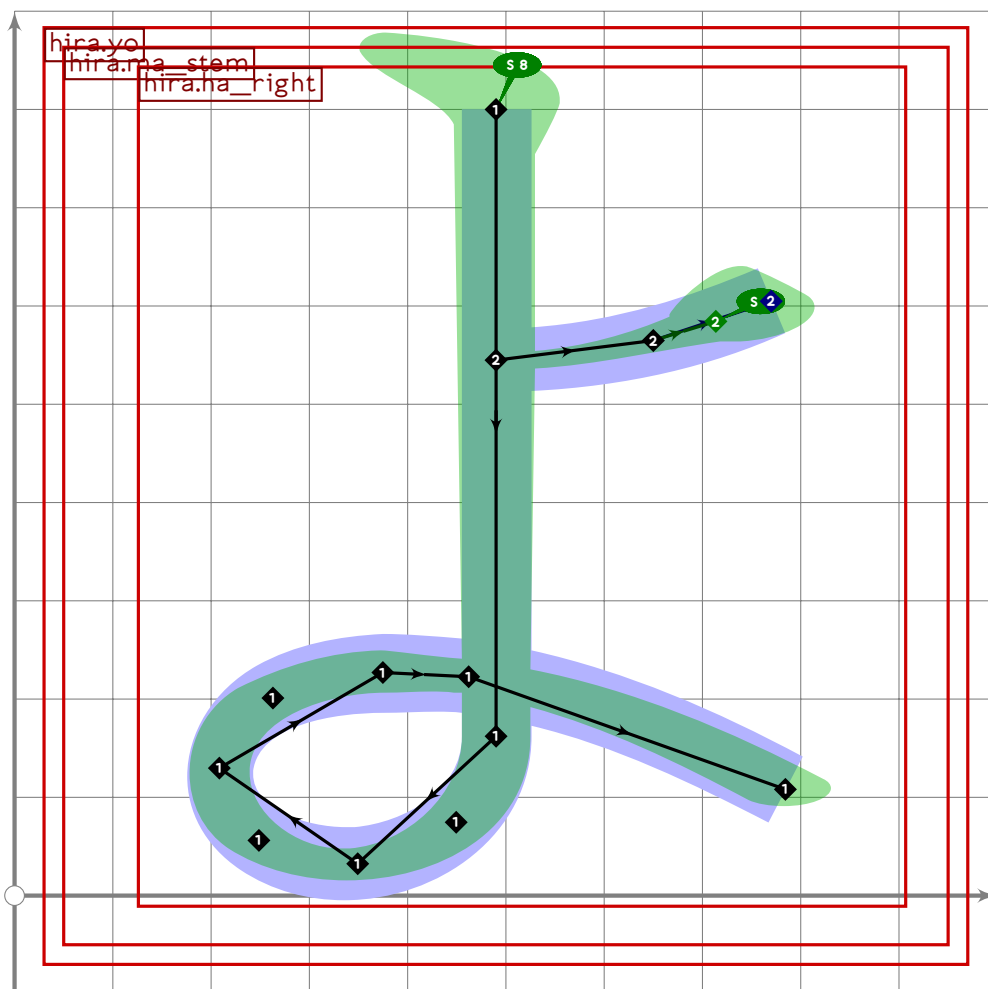
```
723
724 vardef hira.ya =
725   push_pbox_toexpand("hira.ya");
726
727   push_stroke((110,400)..(690,570)..(870,450)..(690,330)..(520,400),
728     (1.8,1.8)–(1.6,1.6)–(1.4,1.4)–(1.8,1.8)–(0.6,0.6));
729   set_boserif(0,0,5);
730
731   push_stroke((575,750)–(620,475),
732     (1.1,1.1)–(1.6,1.6));
733   set_boserif(0,1,4);
734
735   push_stroke((230,680)..tension 2..(420,320)..(570,30),
736     (1.6,1.6)–(1.4,1.4)–(1.7,1.7));
737   set_boserif(0,0,8);
```

```
738 expand_pbox;
739 enddef;
```



```
740
741 vardef hira.yu =
742   push_pbox_toexpand("hira.yu");
743
744   push_stroke((190,710){down}...(170,340)..(190,120),
745     (1.6,1.6)–(1.4,1.4)–(1.5,1.5));
746   set_boserif(0,0,5);
747
748   push_stroke((point 1.7 of get_stroke(0))
749     {-direction 1.7 of get_stroke(0)}..(210,330)..
750     (320,500)..(590,620)..(850,390)..(620,190)..tension 1.3..(320,320),
751     (1.4,1.4)–(1.5,1.5)–(1.4,1.4)–(1.5,1.5)–(1.6,1.6)–
752     (1.6,1.6)–(0.77,0.77));
753
754   push_stroke((580,780){down}...(590,300)..{dir 190}(360,-85),
755     (1.6,1.6)–(1.5,1.5)–(0.6,0.6));
756   set_boserif(0,0,8);
757   expand_pbox;
758 enddef;
```

HIRA



```

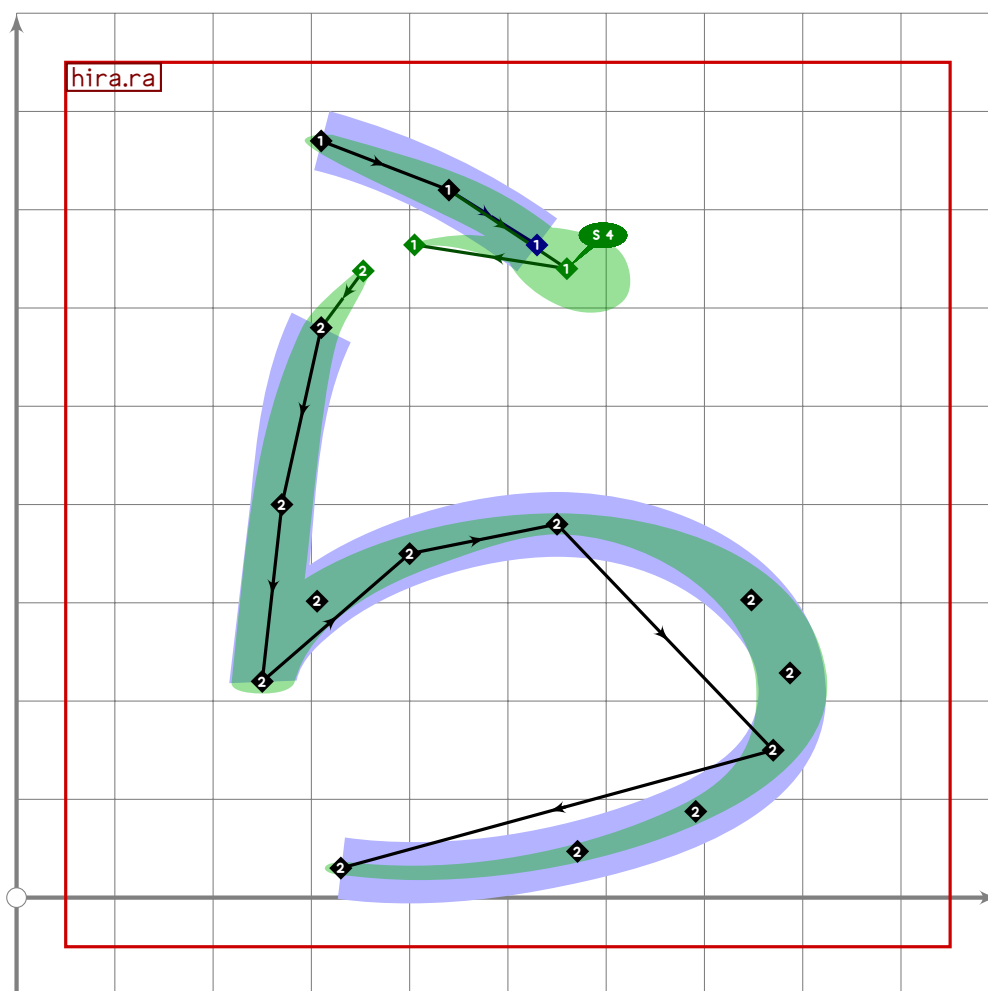
759
760 vardef hira.yo =
761   push_pbox_toexpand("hira.yo");
762
763   hira.ma_stem;
764
765   replace_strokep(0)(oldp shifted (-20,0));
766   z0=point 0.4 of get_strokep(0);
767
768   push_stroke(z0..(z0+(160,20))..(z0+(280,60)+60*mincho*dir 200),
769     (1.2,1.2)-(1.4,1.4)-(1.8,1.8));
770   set_boserif(0,2,6);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

```

HIRA

Hiragana Rarirurero

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

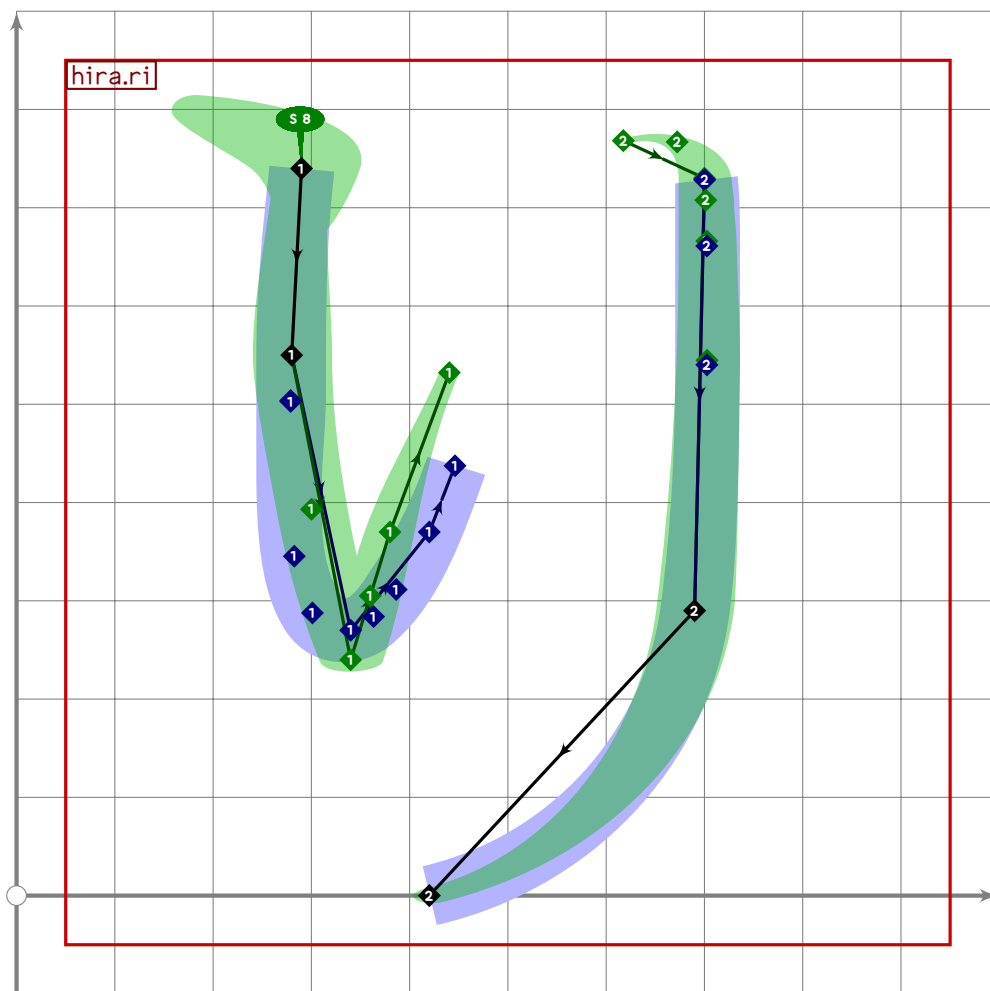


```

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

```

HIRA



```

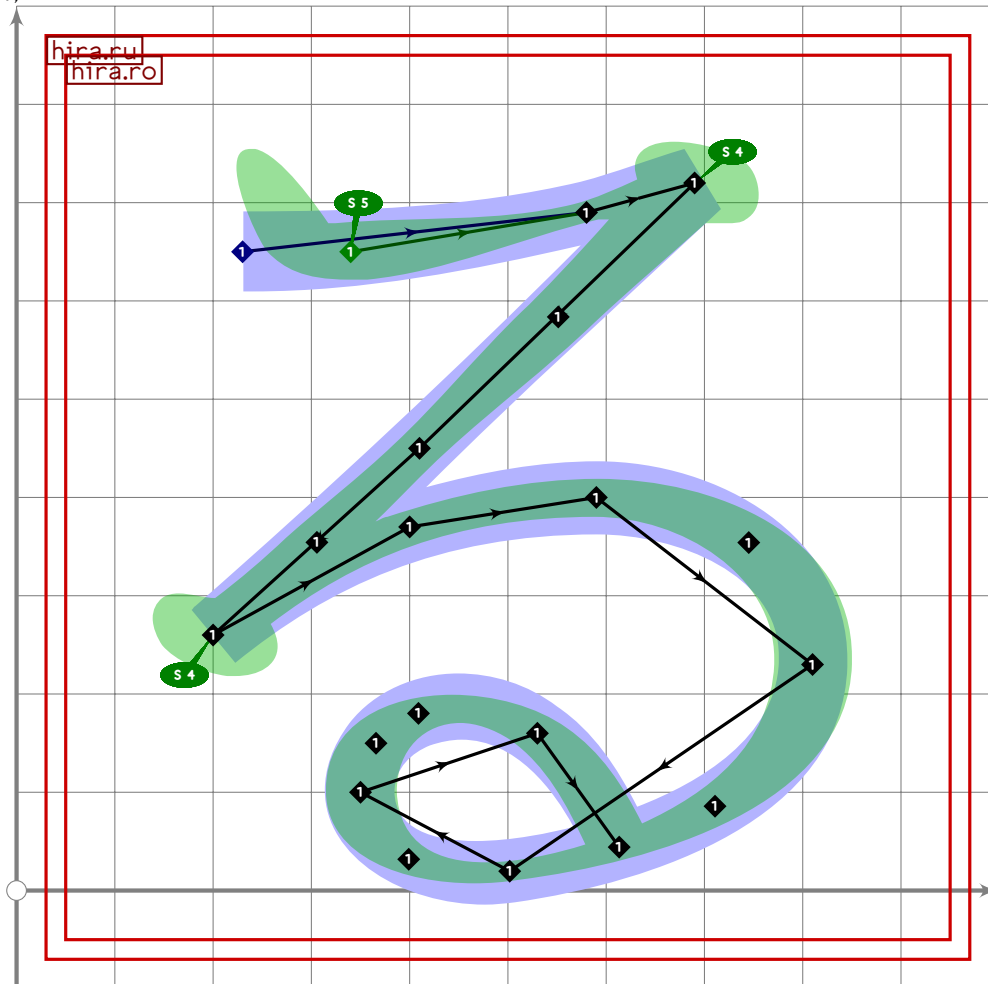
791
792 vardef hira.ri =
793   push_pbox_toexpand("hira.ri");
794
795   begingroup
796     save ripx,ripy,ripz,x,y;
797     path ripx,ripy,ripz;
798     numeric x[],y[];
799     z1=(290,740);
800     z2=(280,550);
801     z3=(340,270-30*mincho);
802     z4=(420-40*mincho,370);
803     z5=(540,710);
804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

```

```

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

```

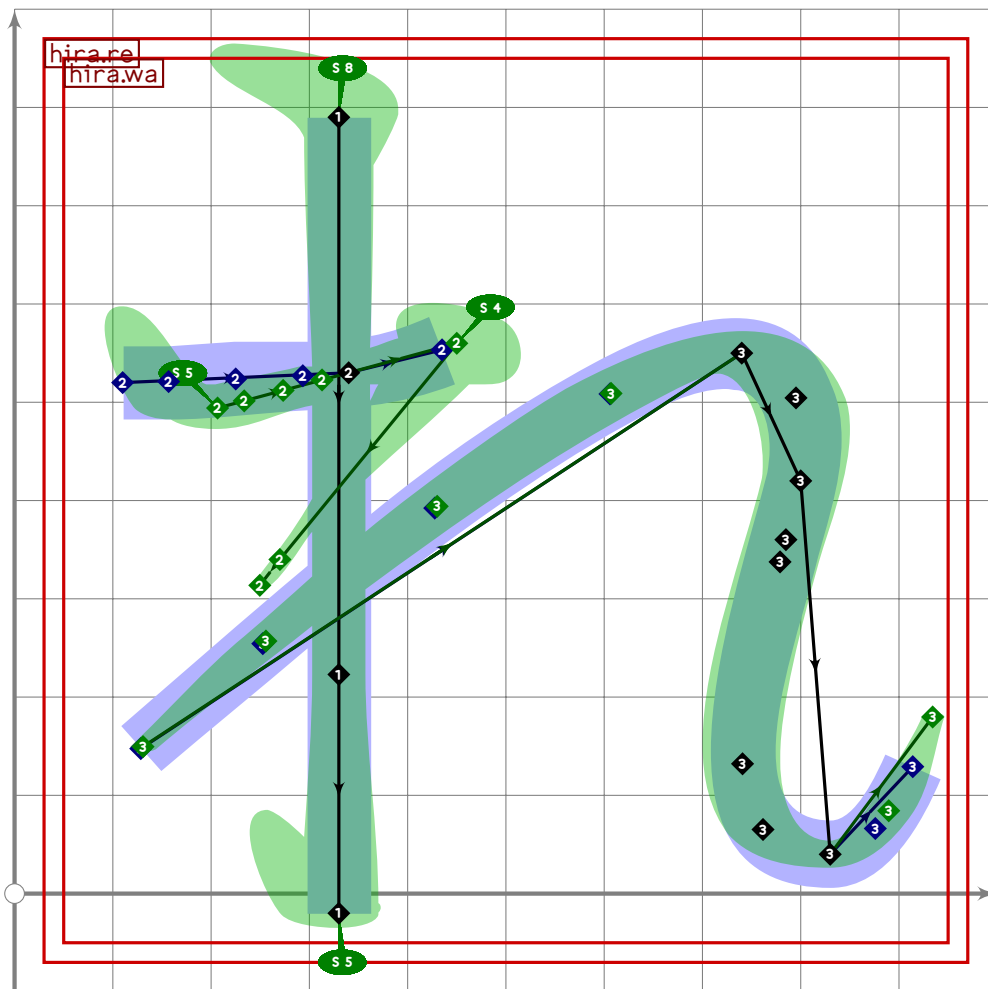


```

818
819  vardef hira.ru =
820    push_pbox_toexpand("hira.ru");
821
822    hira.ro;
823
824    replace_strokep(0)((subpath (0,78) of oldp)..(350,100)..(530,160)..
825      {curl 0.2}(point 7.6 of oldp));
826    replace_strokeq(0)((2.6,2.6)–(1.2,1.2)–(1.9,1.9)–
827      (1.3,1.3)–(1.6,1.6)–
828      (1.5,1.5)–(1.9,1.9)–(1.6,1.6)–
829      (1.2,1.2)–(1.5,1.5)–(1.4,1.4));
830    expand_pbox;
831  enddef;

```

HIRA

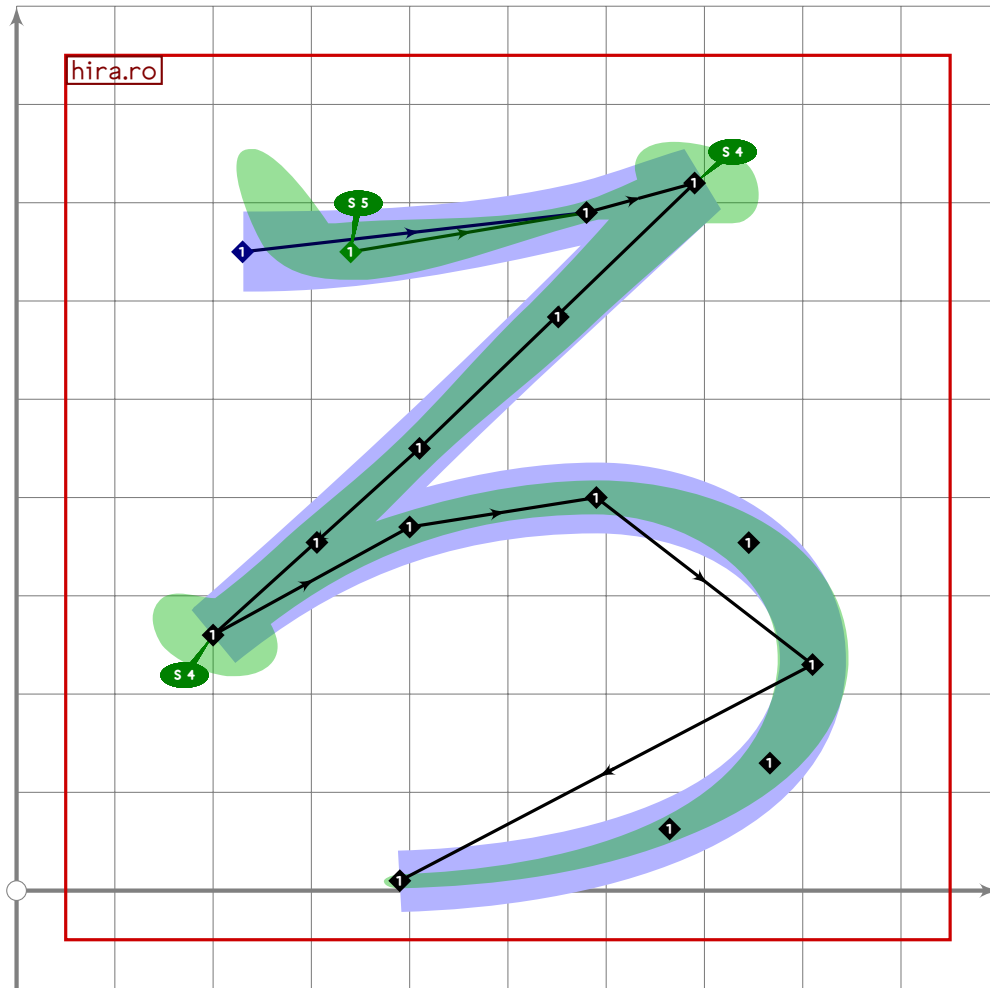


```

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

```

HIRA



```

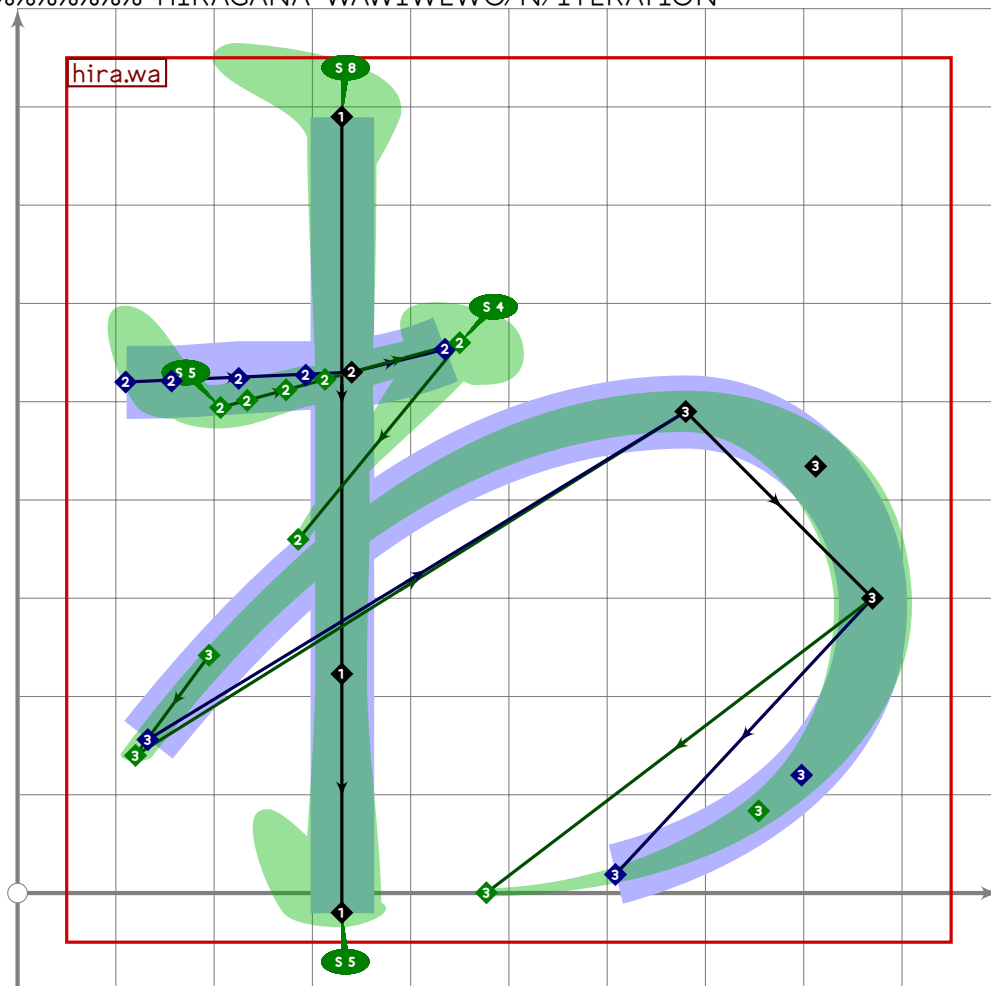
847
848 vardef hira.ro =
849   push_pbox_toexpand("hira.ro");
850
851   push_stroke((230+110*mincho,650)..(580,690)..{curl 1}(690,720){curl 1}..
852     (410,450)..{curl 1}(200,260){curl 1}..
853     (400,370)..(590,400){right}..(810,230)..
854     tension 1.1..{curl 0}(390,10),
855     (2.6,2.6)-(1.2,1.2)-(1.9,1.9)-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

```

HIRA

Hiragana Wawiwewo/N/Iteration

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



```

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

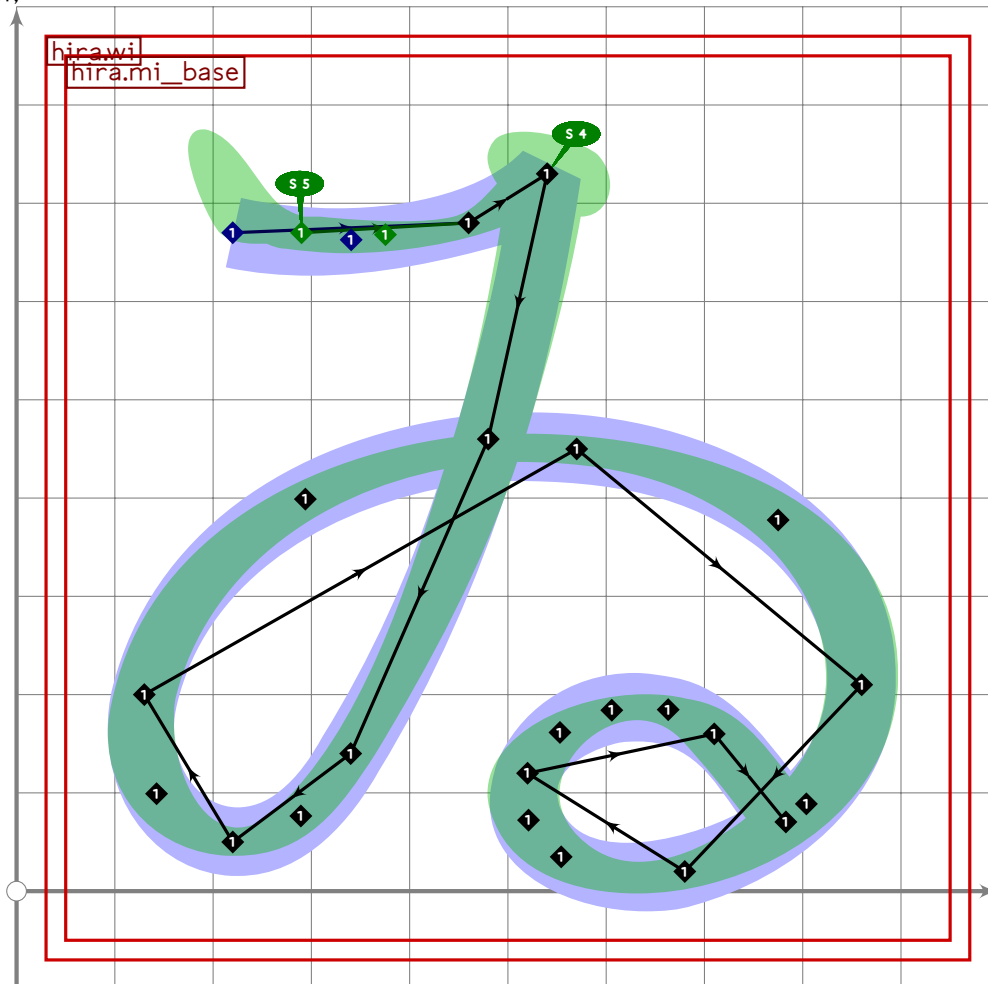
```

HIRA

```

885 set_boserif(0,0,5);
886 set_boserif(0,2,4);
887 expand_pbox;
888 enddef;

```

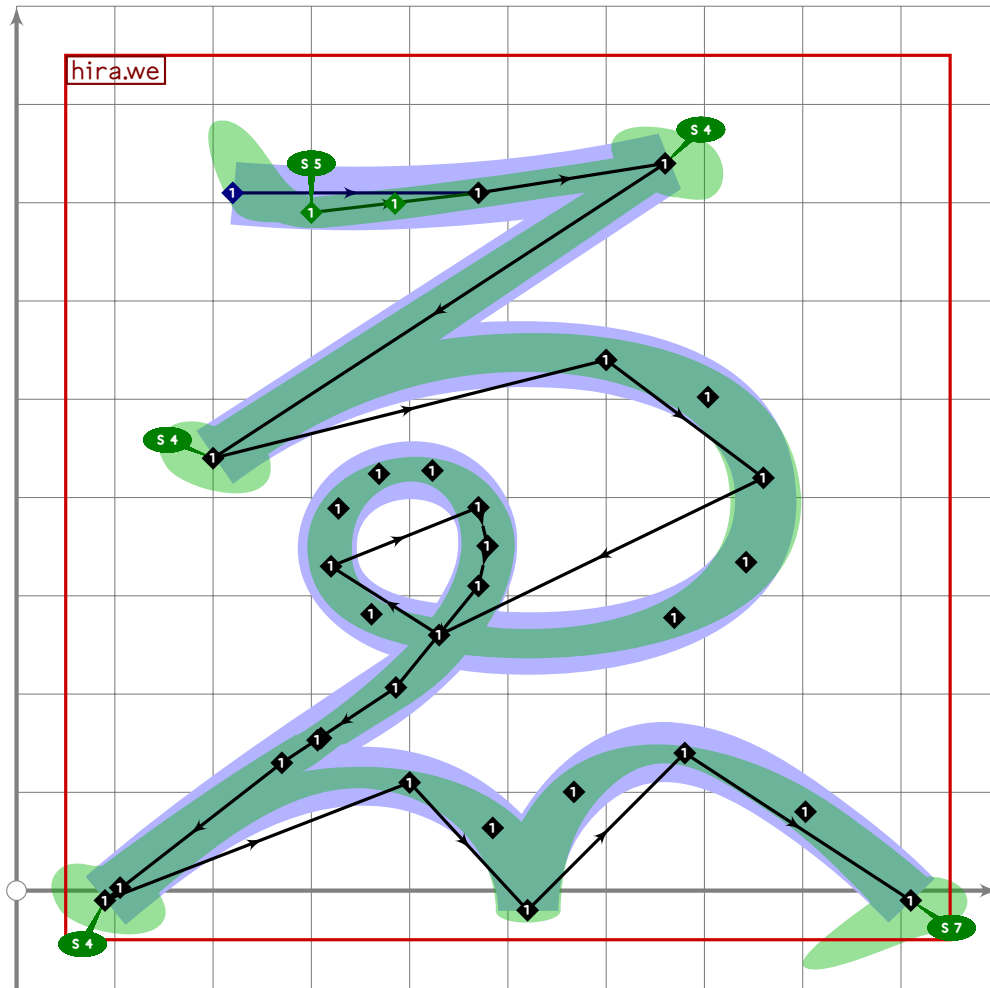


```

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

```

HIRA



```

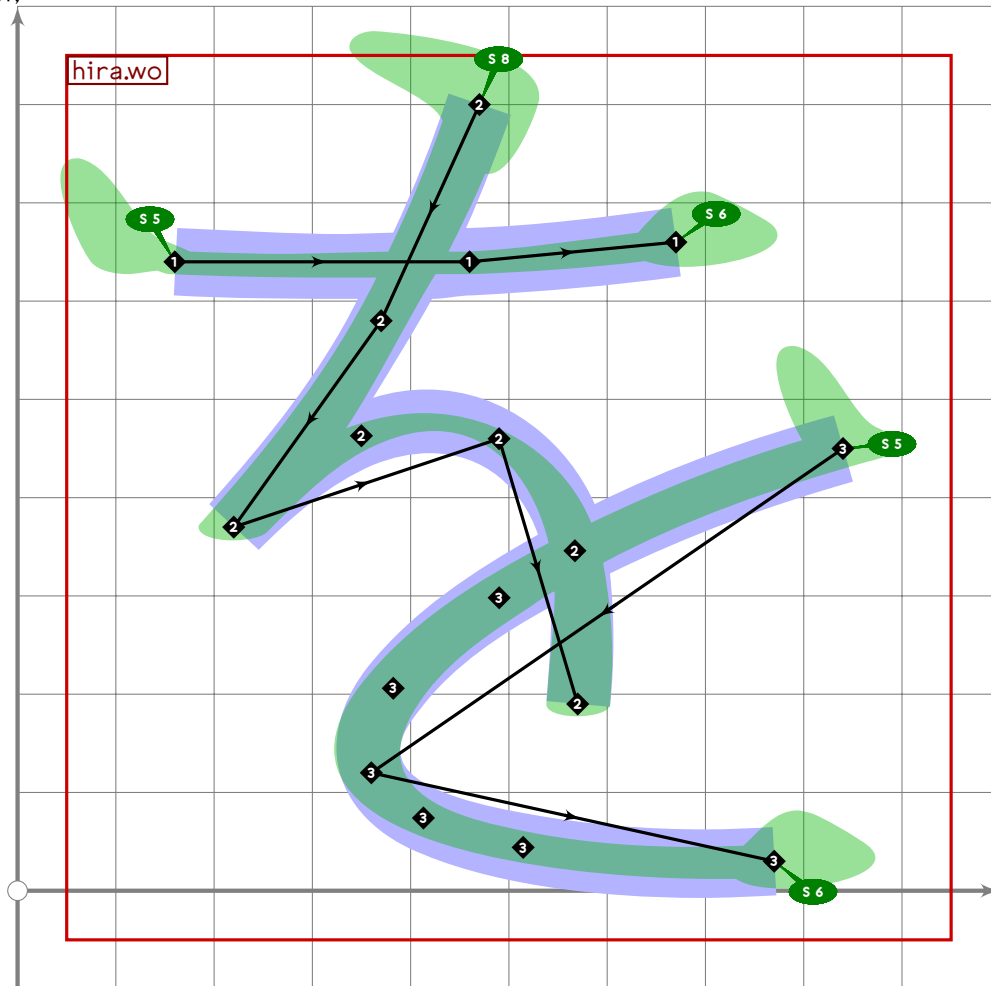
904
905 vardef hira.we =
906   push_pbox_toexpand("hira.we");
907
908   push_stroke((220+80*mincho,710-20*mincho)..(470,710)..{curl 1}(660,740)-
909     (200,440){curl 1}..
910     (600,540)..(760,420)..(430,260)..(320,330)..(470,390)..(470,310)..
911     (270,130)..{curl 0}(90,-10){curl 0.1}..
912 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

```

```

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

```



```

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

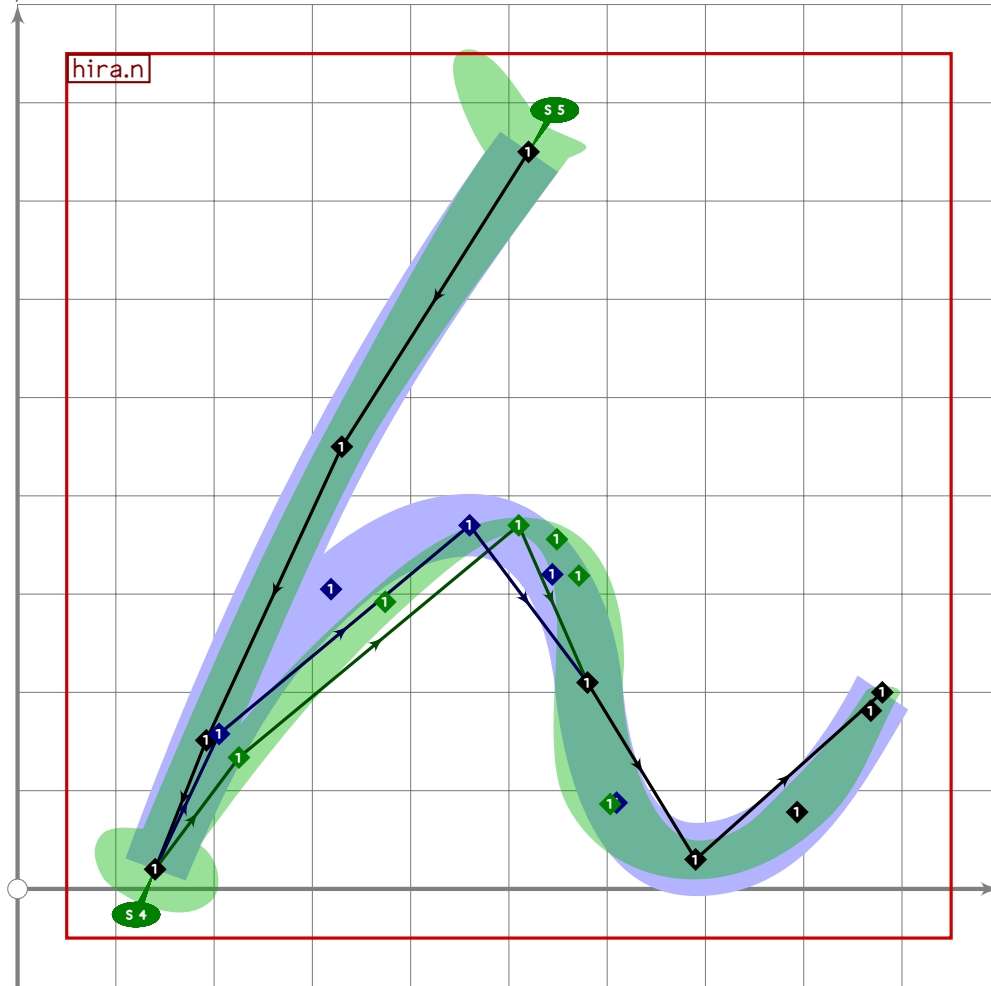
```

HIRA


```

946
947 push_stroke((840,450){curl 0.017}..tension 1 and 2..(360,120)..
948   tension 2 and 1..{curl 0.03}(770,30),
949   (1,1,7)-(1,4,14)-(1,7,1,7));
950 set_boserif(0,0,5);
951 set_boserif(0,2,6);
952 expand_pbox;
953 enddef;

```



HIRA

```

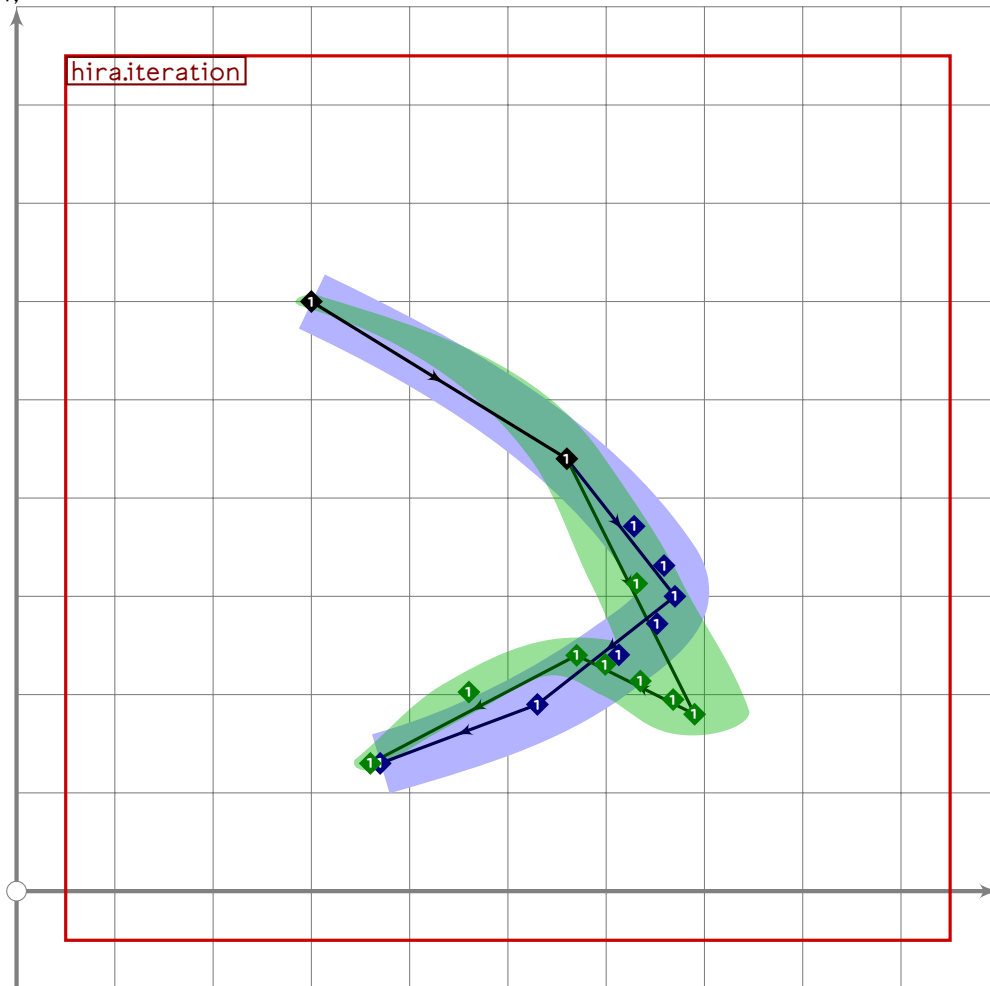
954
955 vardef hira.n =
956   push_pbox_toexpand("hira.n");
957
958   push_stroke((520,750)..(330,450)..{curl 0.2}(140,20){curl 0.1}..
959     tension (1.2+0.6*mincho)..(460+50*mincho,370){right}..
960     (580,210)..(690,30){right},
961     (1,1,7)-(1,2,1,2)-(1,3,1,3)-
962     (1,1,1)-(1,5,1,5)-(1,9,1,9)-(1,1));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

```

```

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

```



```

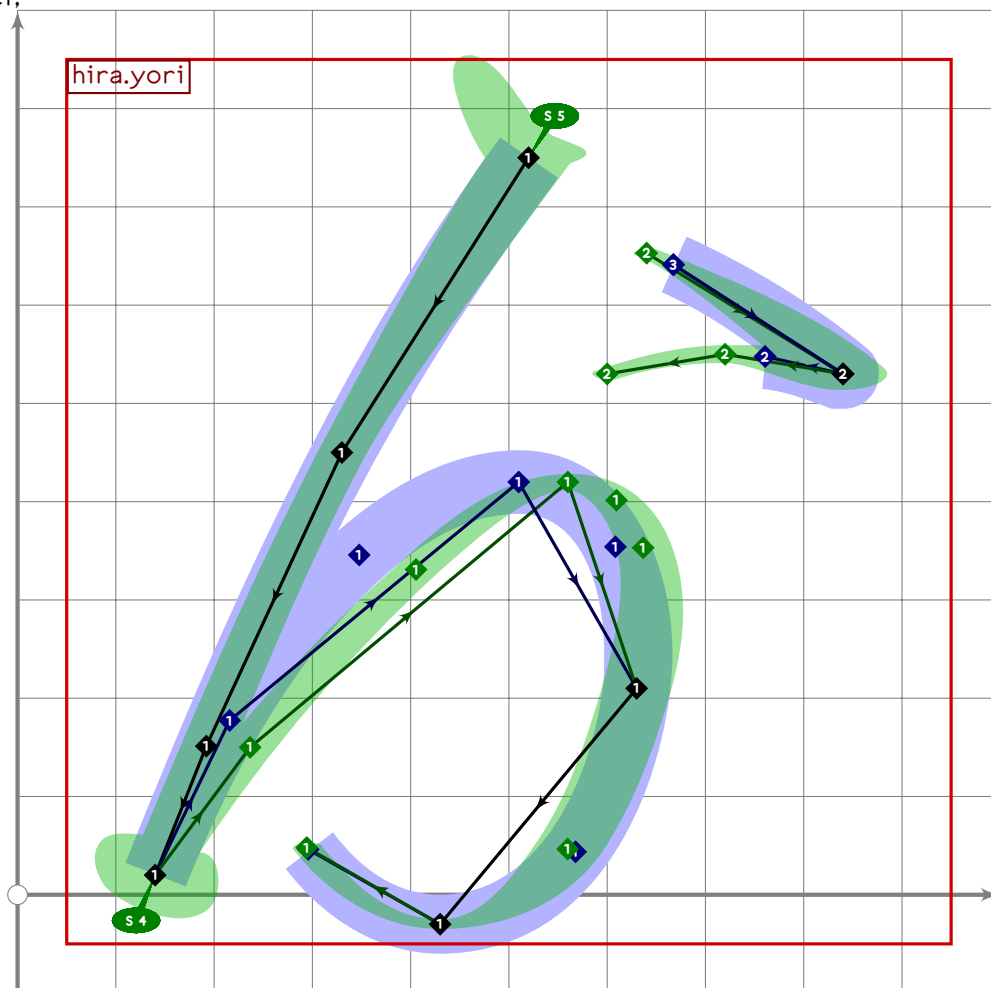
971
972 vardef hira.iteration =
973   push_pbox_toexpand("hira.iteration");
974
975   push_stroke(begingroup
976     save ripx,ripy;
977     path ripx,ripy;
978     ripx:=(300,600){curl 0.2}..(560,440)..
979       tension 1.5 and 2..(670,300)..
980       tension 2 and 1.5..(530,190)..{curl 0.2}(370,130);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

```

HIRA

U+309F
tsuku.uni309F

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



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

HIRA

1009 enddef;

iching.mp

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

---


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

```

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

```

katakana.mp

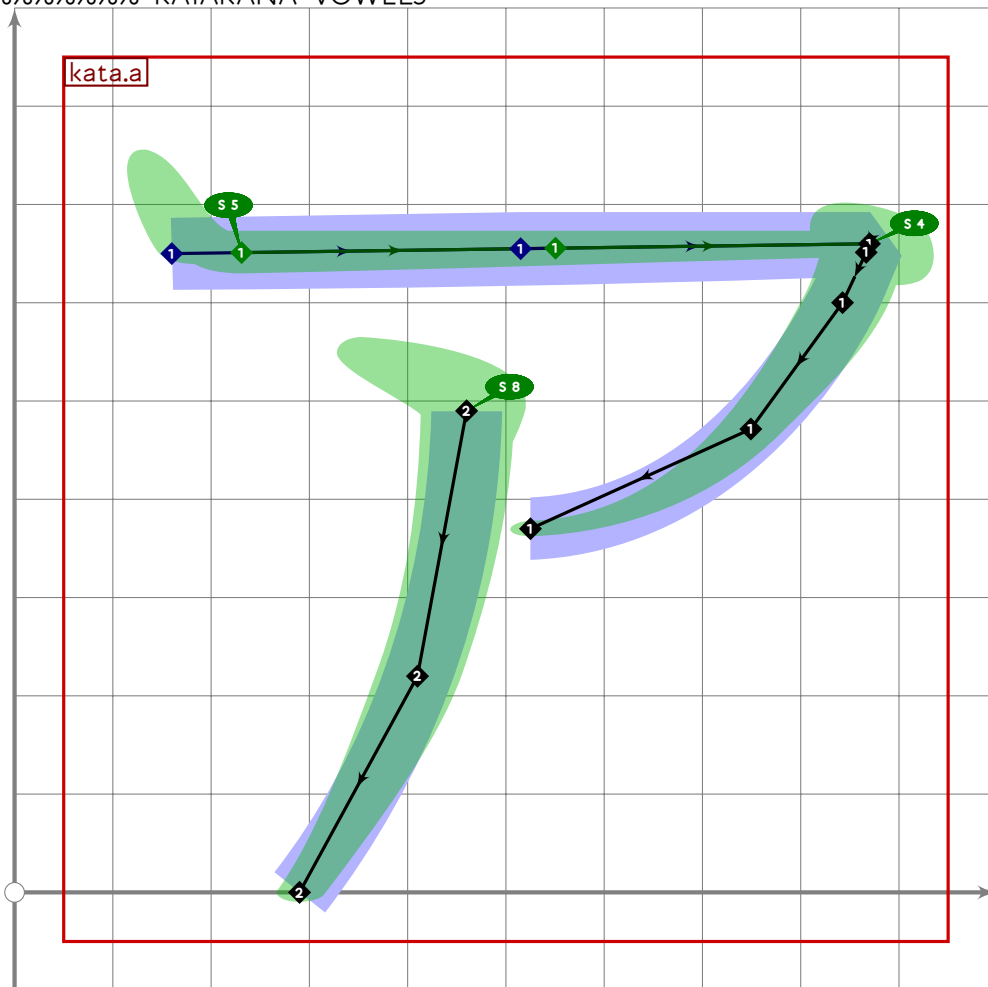
```

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

```

Katakana Vowels

35 %%%%%%%%%%%%%%%%% KATAKANA VOWELS



```

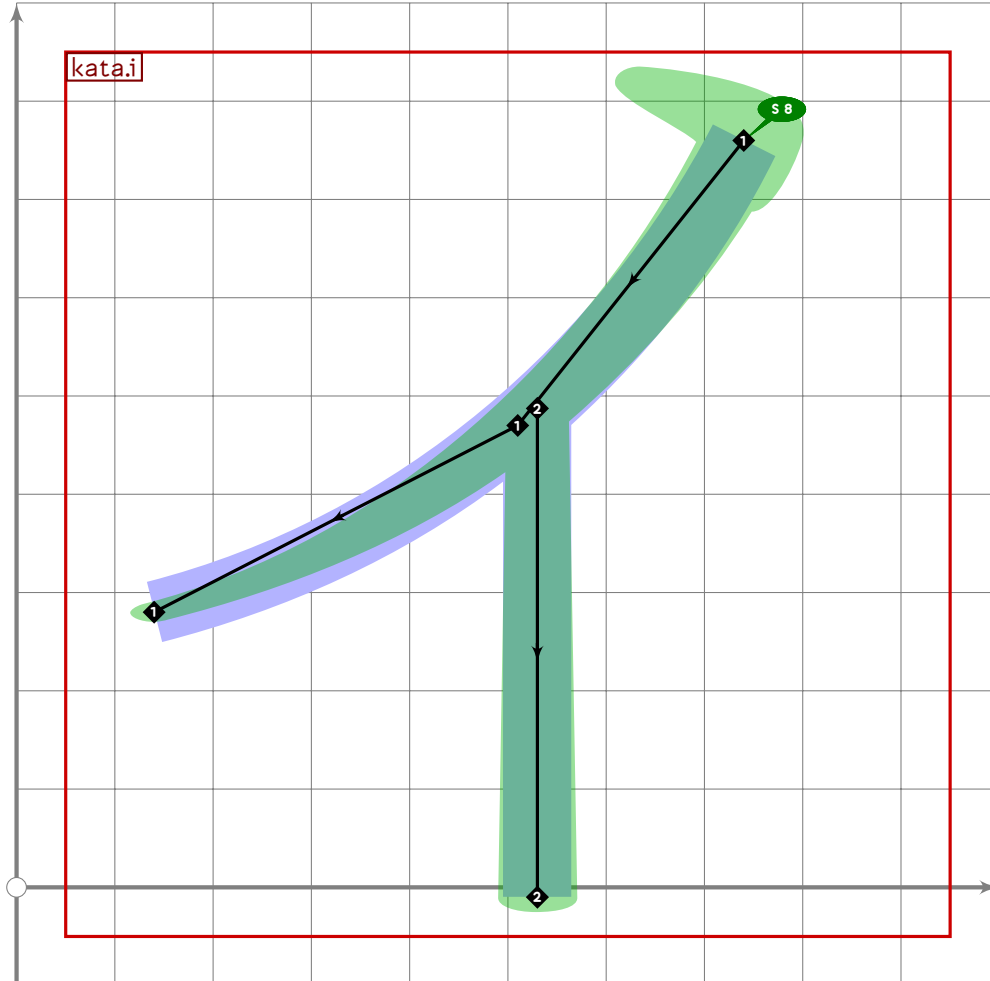
36
37 vardef kata.a =
38   push_pbox_toexpand("kata.a");
39
40   kata.fu_stroke((160,650),(870,660),(525,370));
41

```

```

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;

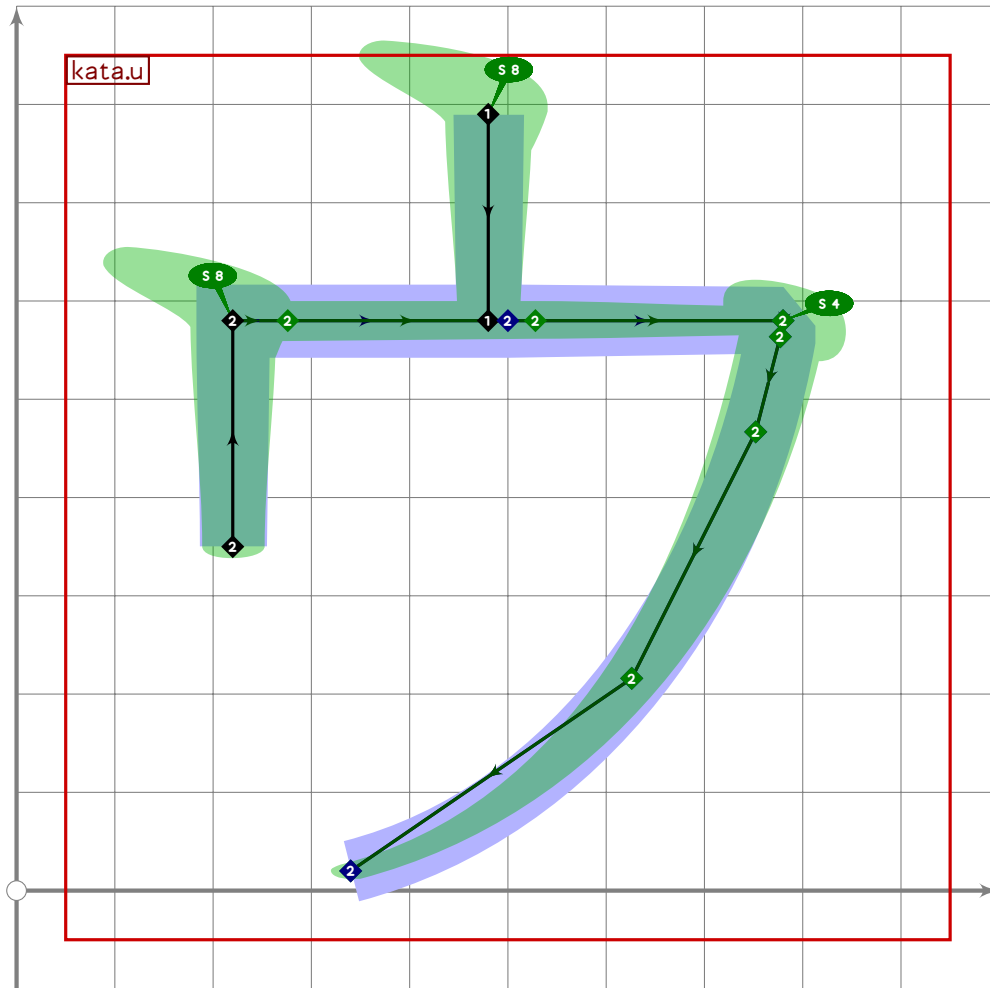
```



```

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;

```

```

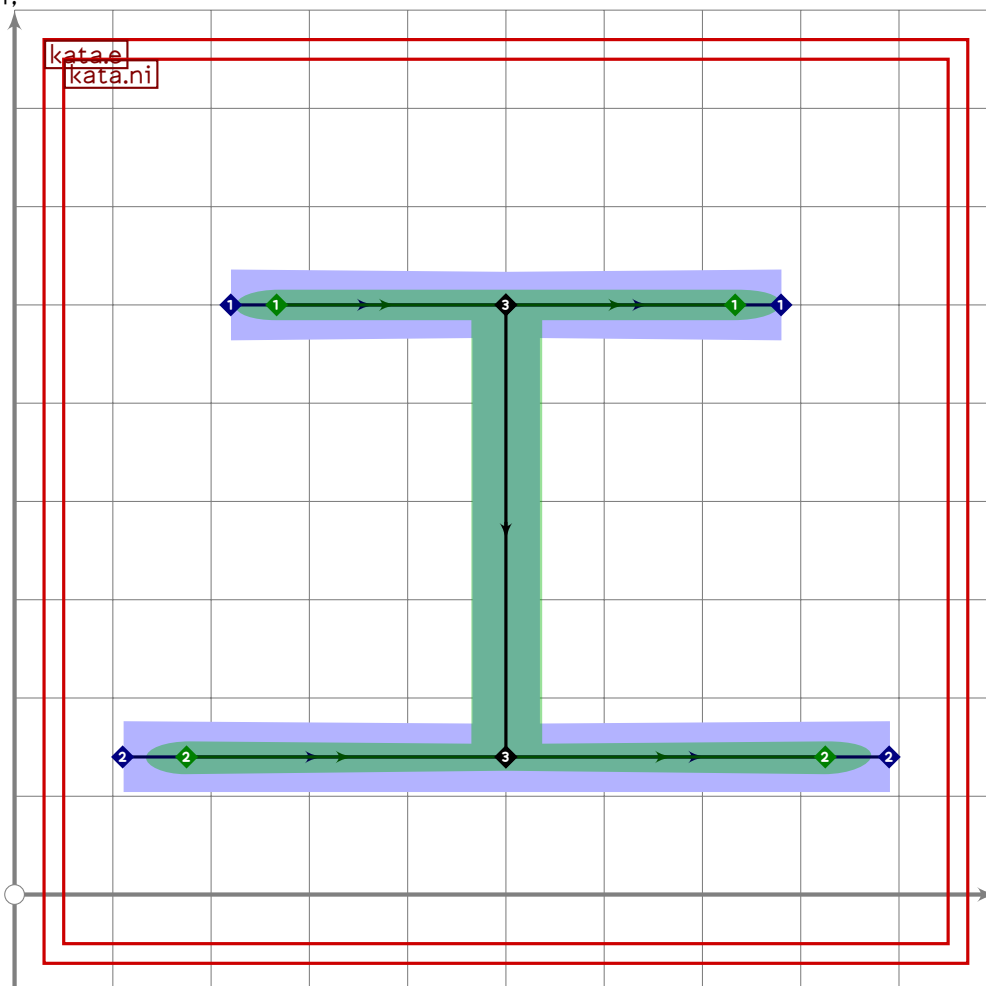
60
61 vardef kata.u =
62   push_pbox_toexpand("kata.u");
63
64   push_stroke((480,790)–(480,580),
65     (1.7,1.7)–(1.4,1.4));
66   set_boserif(0,0,8);
67
68   kata.fu_stroke((220,580),(780,580),(340,20));
69   if mincho>0.01:
70     replace_strokep(0)((220,350)–(220,580)–oldp);
71     replace_strokeq(0)((1.4,1.4)–(1.7,1.7)–oldq);
72     set_botip(0,2,1);
73     set_botip(0,4,0);
74     set_boserif(0,2,whatever);
75     set_boserif(0,4,4);
76   else:
77     replace_strokep(0)((220,350)–oldp);
78     replace_strokeq(0)((1.4,1.4)–oldq);
79     set_botip(0,1,1);
80     set_botip(0,3,0);

```

```

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

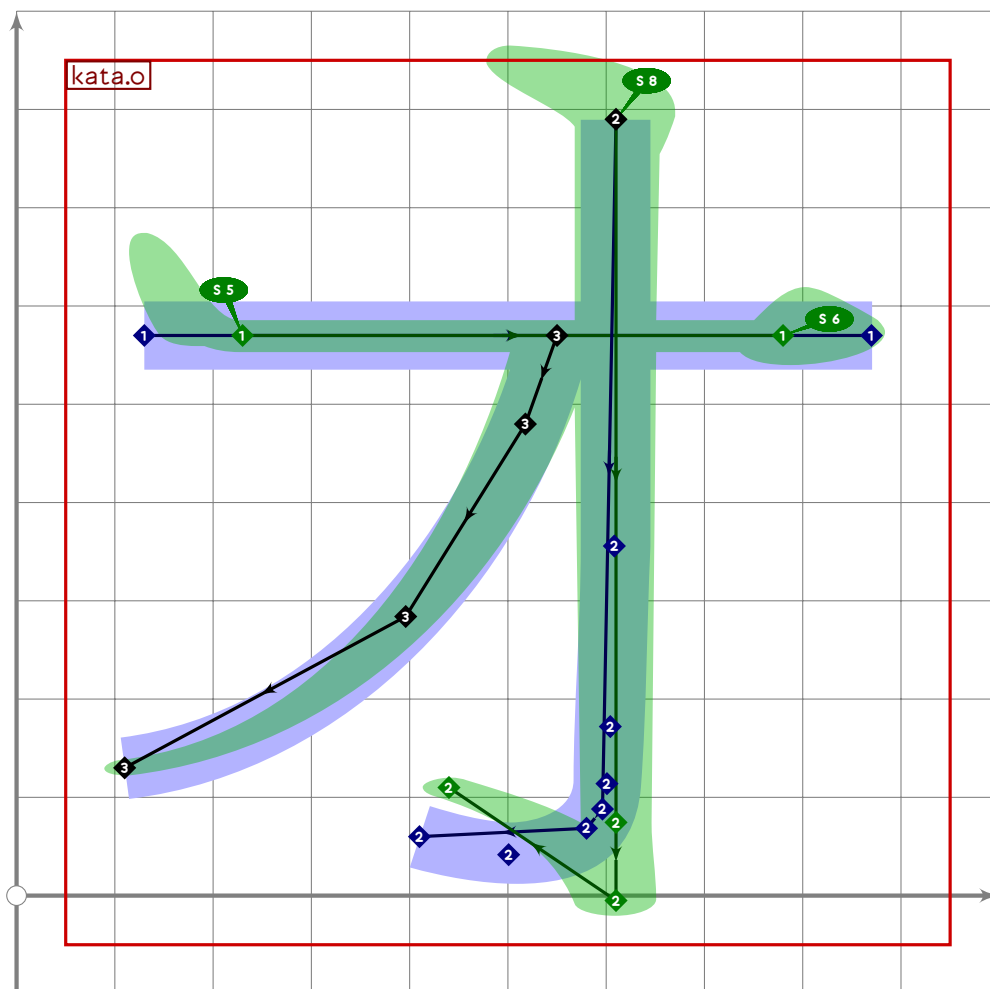
```



```

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

```



```

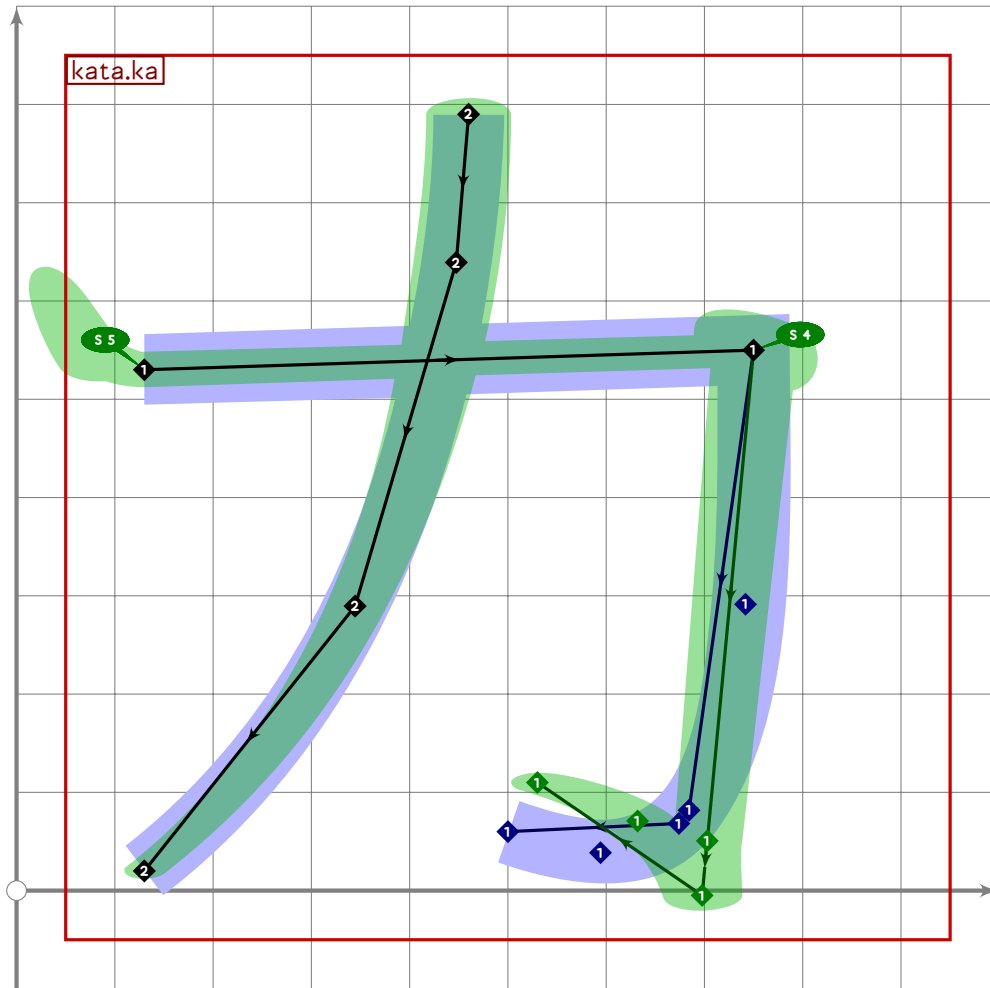
97
98 vardef kata.o =
99   push_pbox_toexpand("kata.o");
100
101   push_stroke((130+100*mincho,570)–(870–90*mincho,570),
102     (1.8,1.8)–(1.6,1.6));
103   set_boserif(0,0,5);
104   set_boserif(0,1,6);
105
106   kata.ho_centre((610,790),(610,20));
107
108   kata.no_stroke((550,570),(110,130));
109 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

```

KATA

Katakana Kakikukeko/Gagigugego

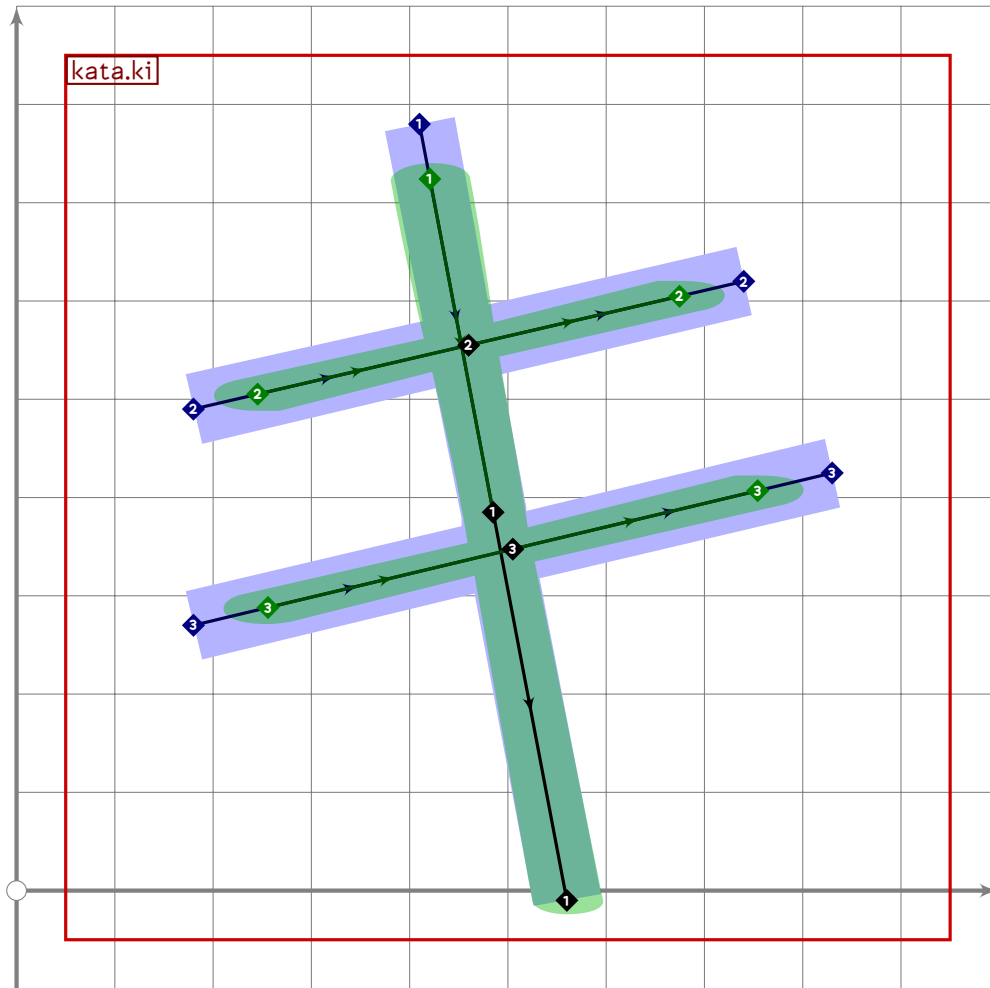
112 %%%%%%%%% KATAKANA KAKIKUKEKO/GAGIGUGEGO



```

113
114 vardef kata.ka =
115   push_pbox_toexpand("kata.ka");
116
117   kata.ho_centre((750,550),(700,20));
118
119   replace_strokep(0)((130,530)-oldp);
120   replace_strokeq(0)((1,8,1,8)-oldq);
121   set_botip(0,1,1);
122   set_botip(0,2,whatever);
123   set_botip(0,3,0);
124   set_boserif(0,0,5);
125   set_boserif(0,1,4);
126
127   kata.no_stroke((460,790),(130,20));
128   expand_pbox;
129 enddef;

```

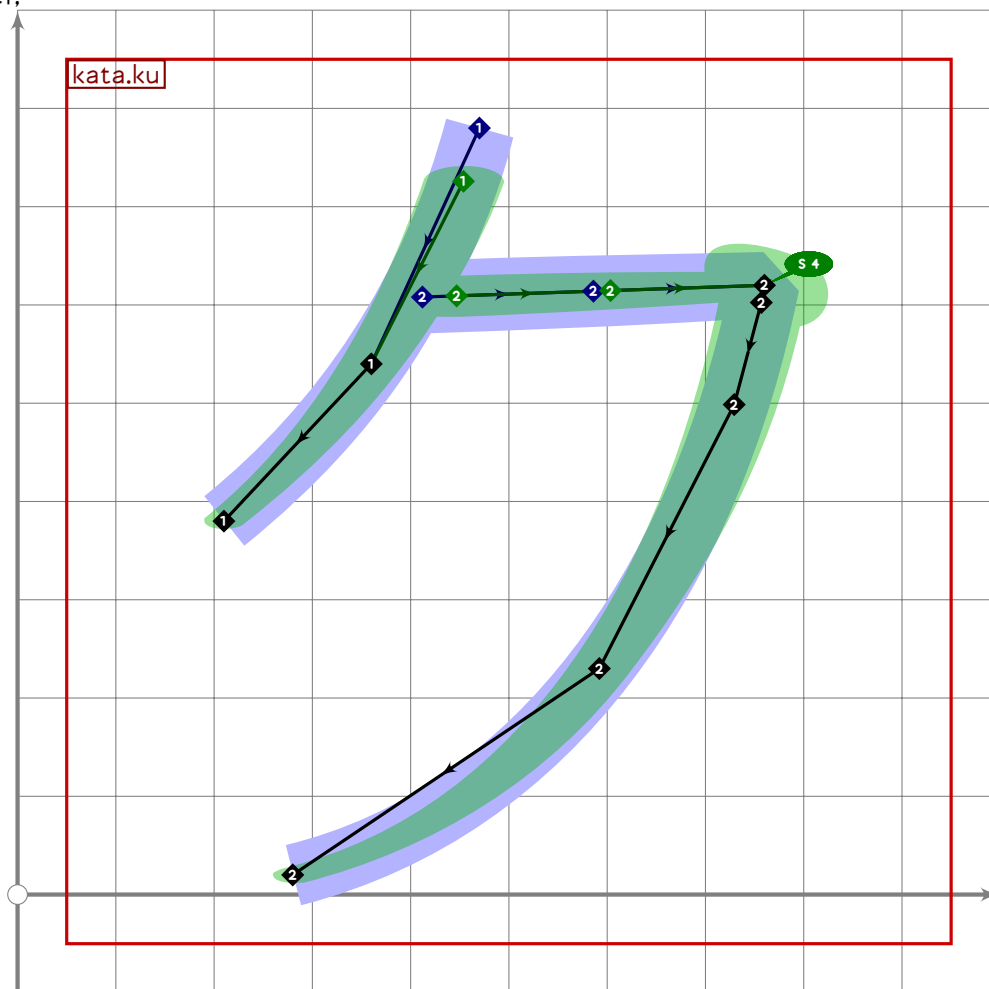


```

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

```

151 enddef;



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;

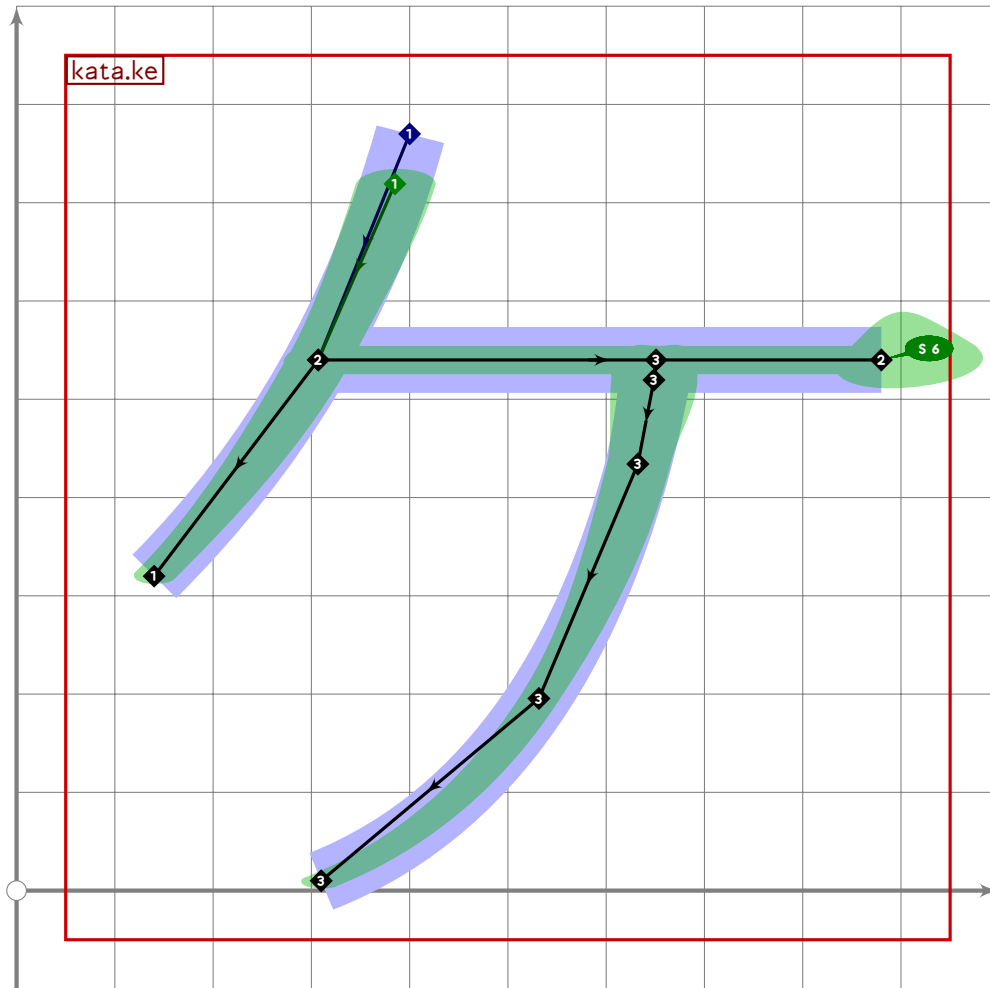
161 kata.fu_stroke(z1,(760,620),(280,20));

162 set_boserif(0,0,whatever);

163 expand_pbox;

164 enddef;

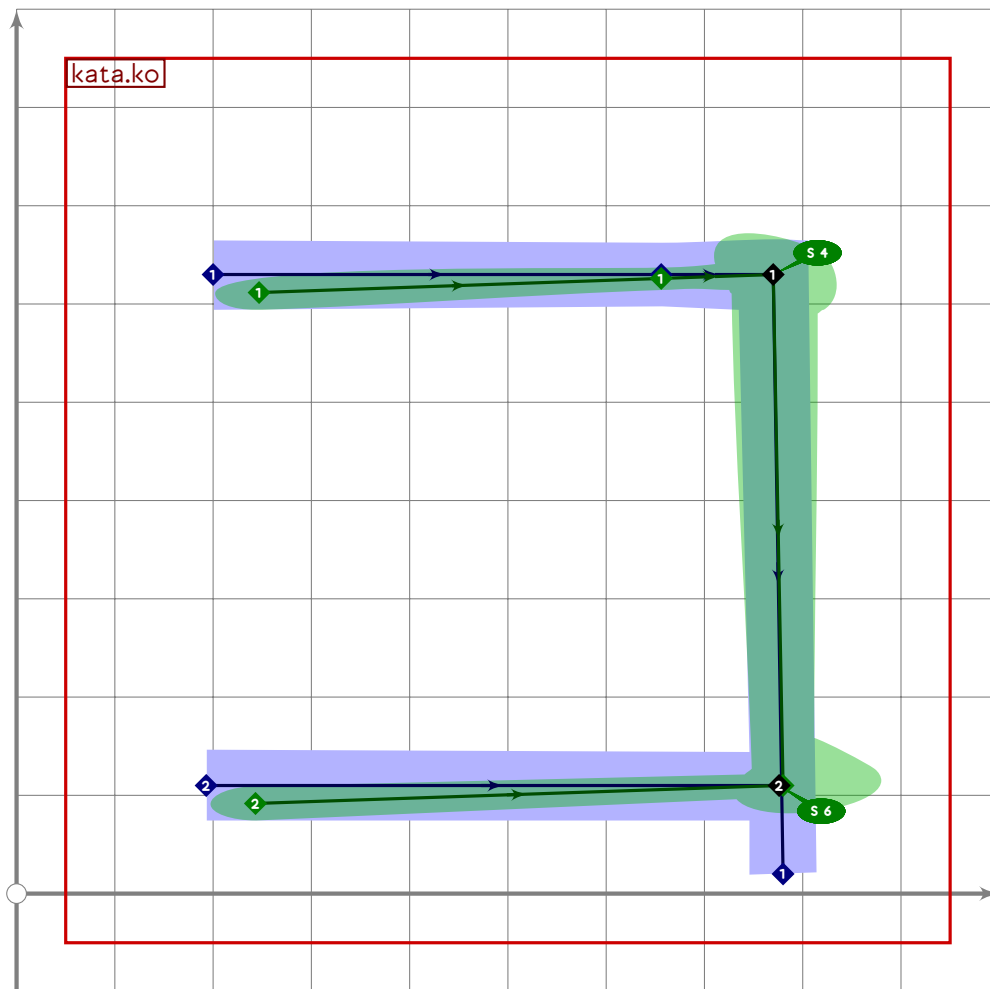
KATA



```

165
166 vardef kata.ke =
167   push_pbox_toexpand("kata.ke");
168
169   push_stroke((400,770)..(307,540)..(140,320),
170     (0.68,2.7)-(1.4,1.4)-(1.1,1.1));
171   set_boserif(0,0,5);
172
173   z1=(get_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;

```



```

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

```

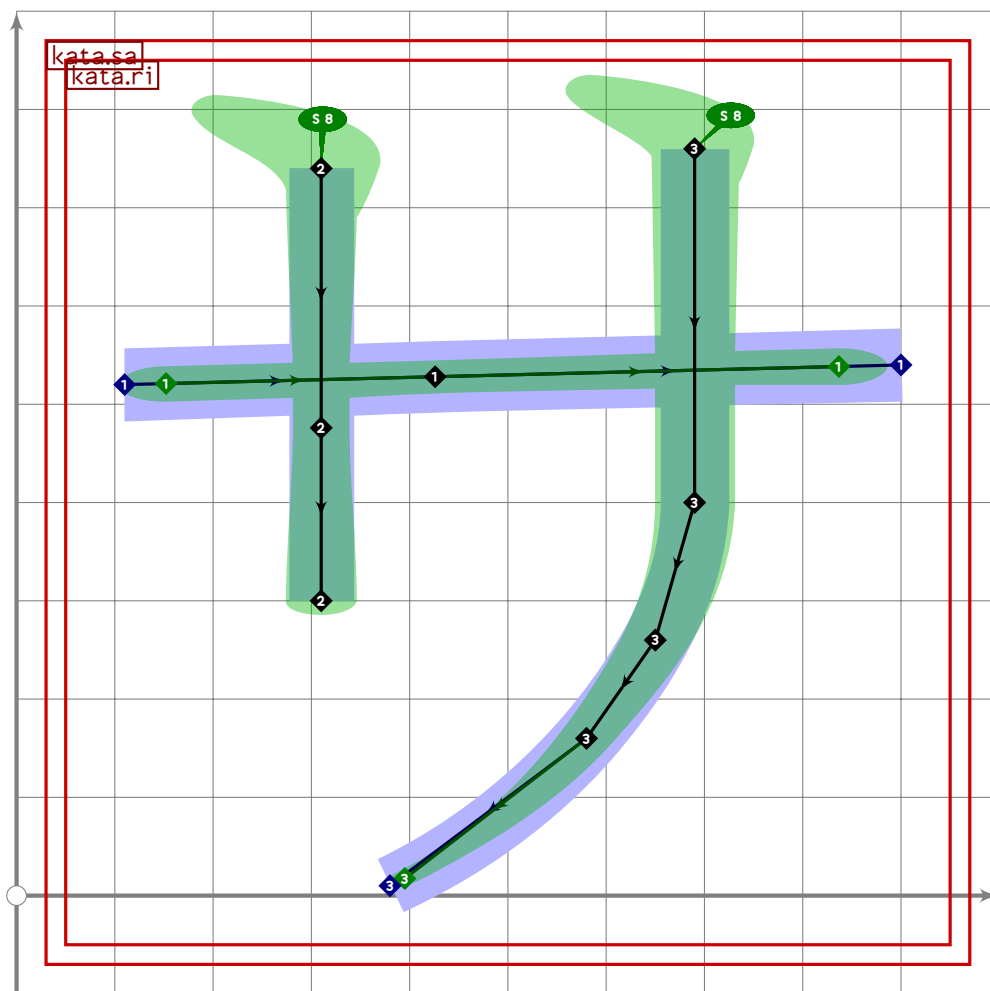
KATA

Katakana Sashisuseso/Zajizuzezo

```

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

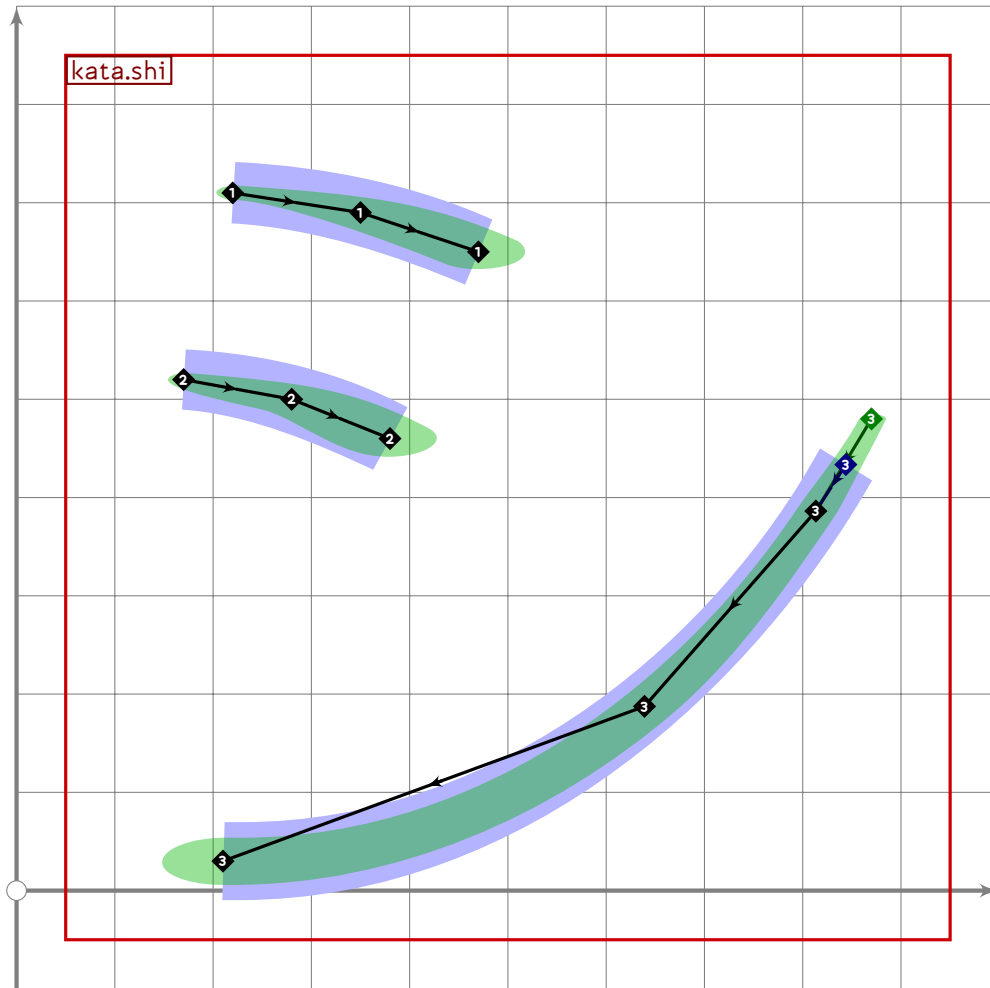
```

```

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

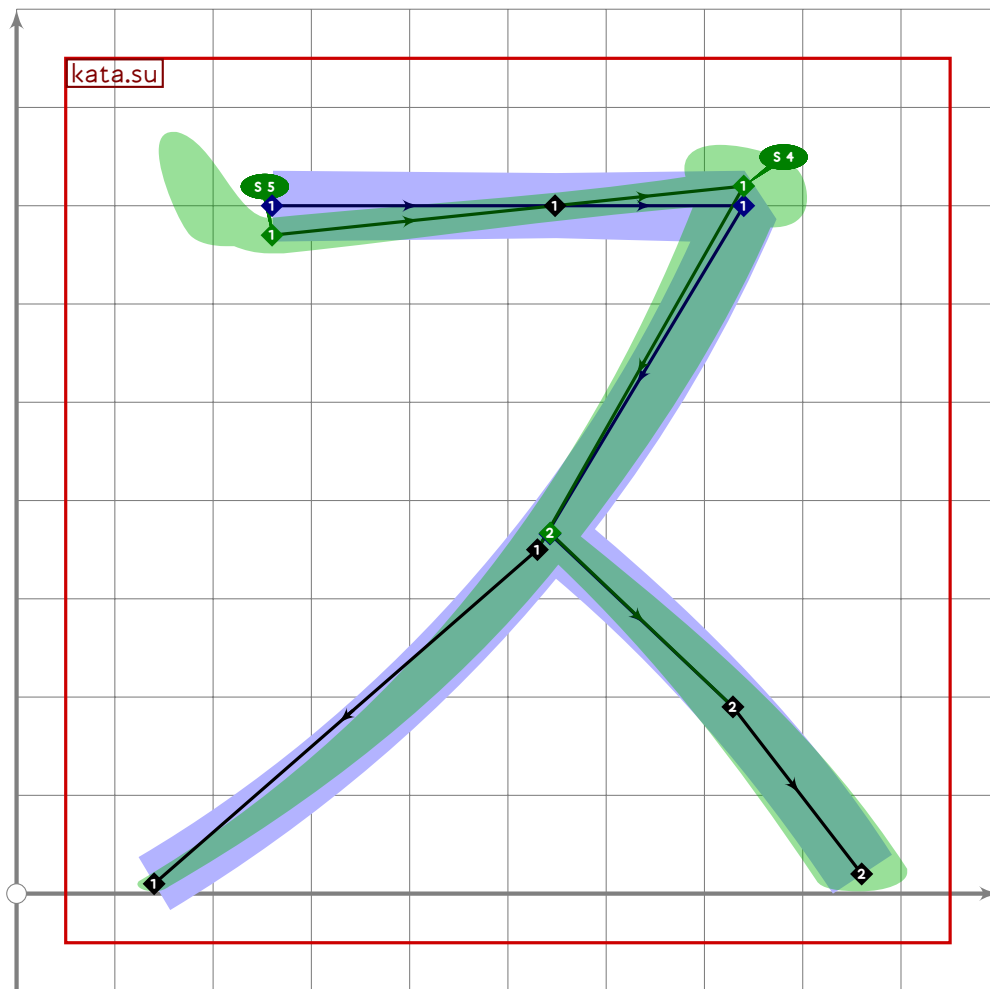
```



```

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

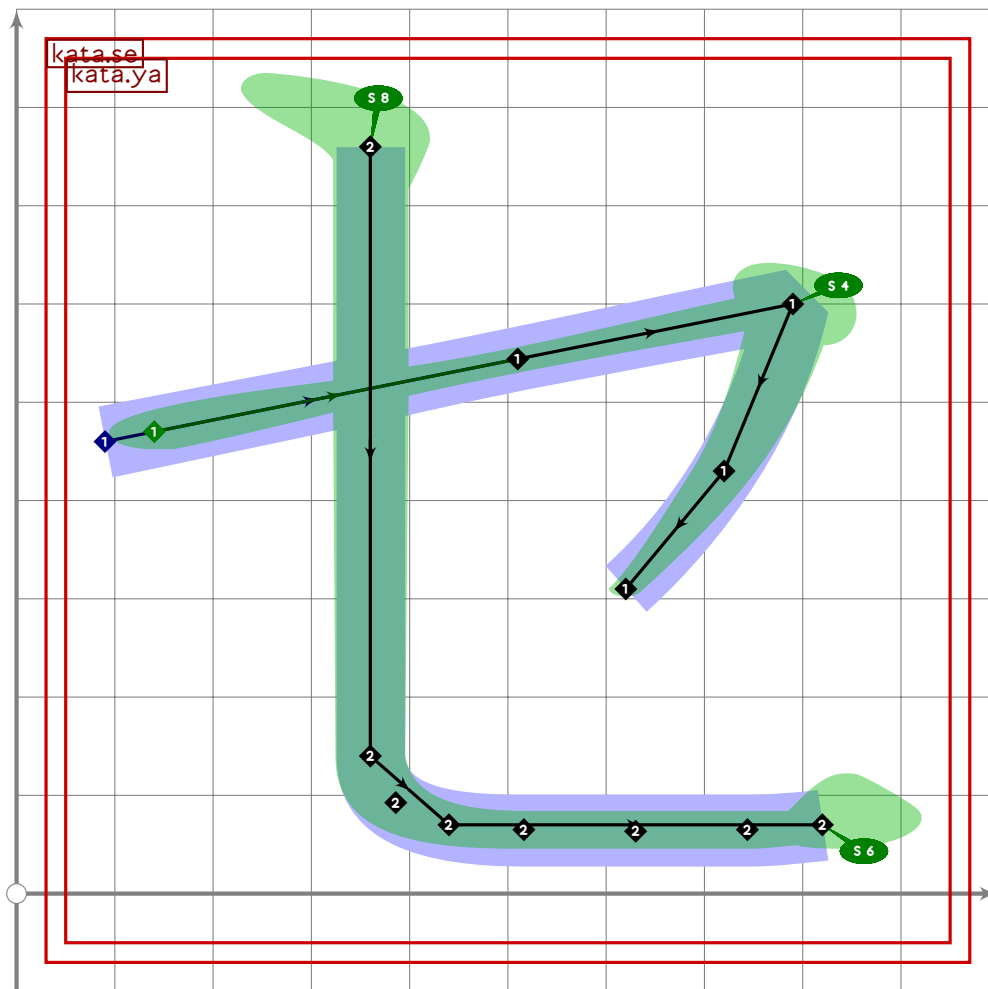
```



```

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_strokep(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_strokep(0))..(729,190)..(860,20),
240     (1.2,1.2)-(1.6,1.6)-(1.8,1.8));
241   expand_pbox;
242 enddef;

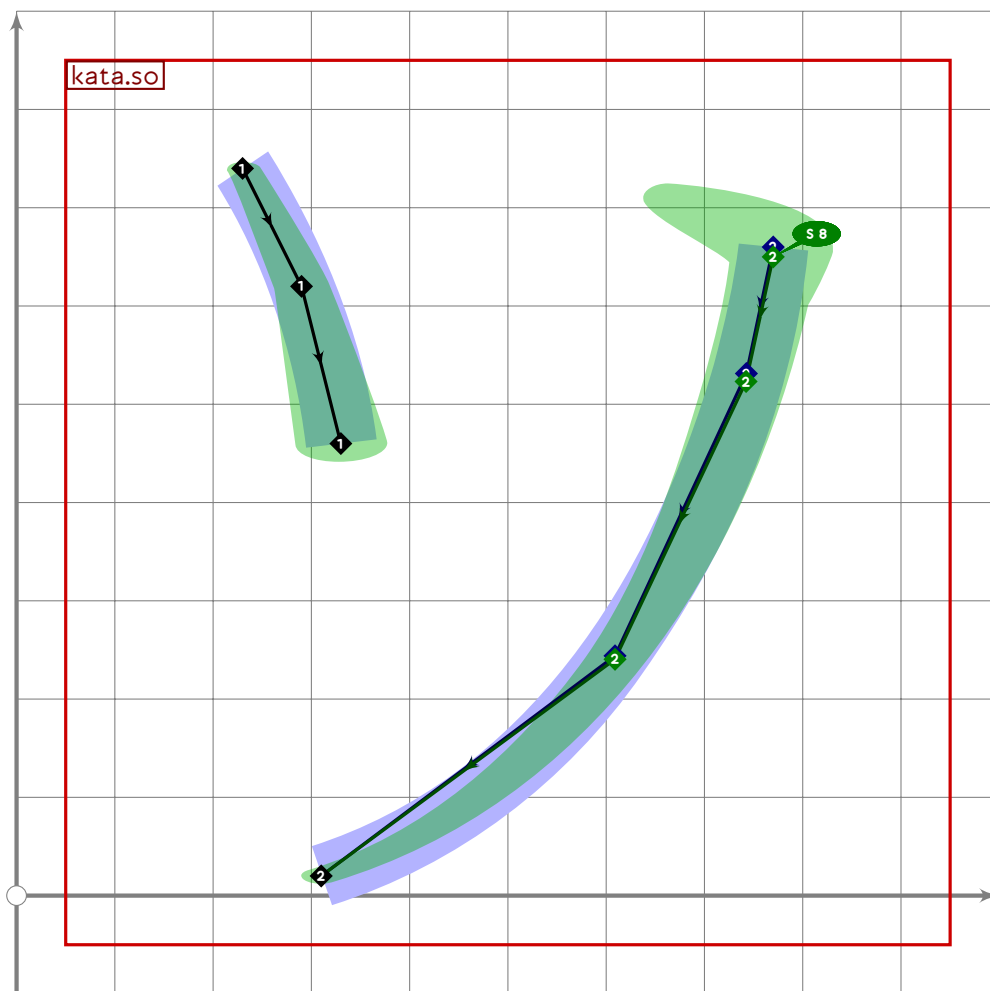
```



```

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

```

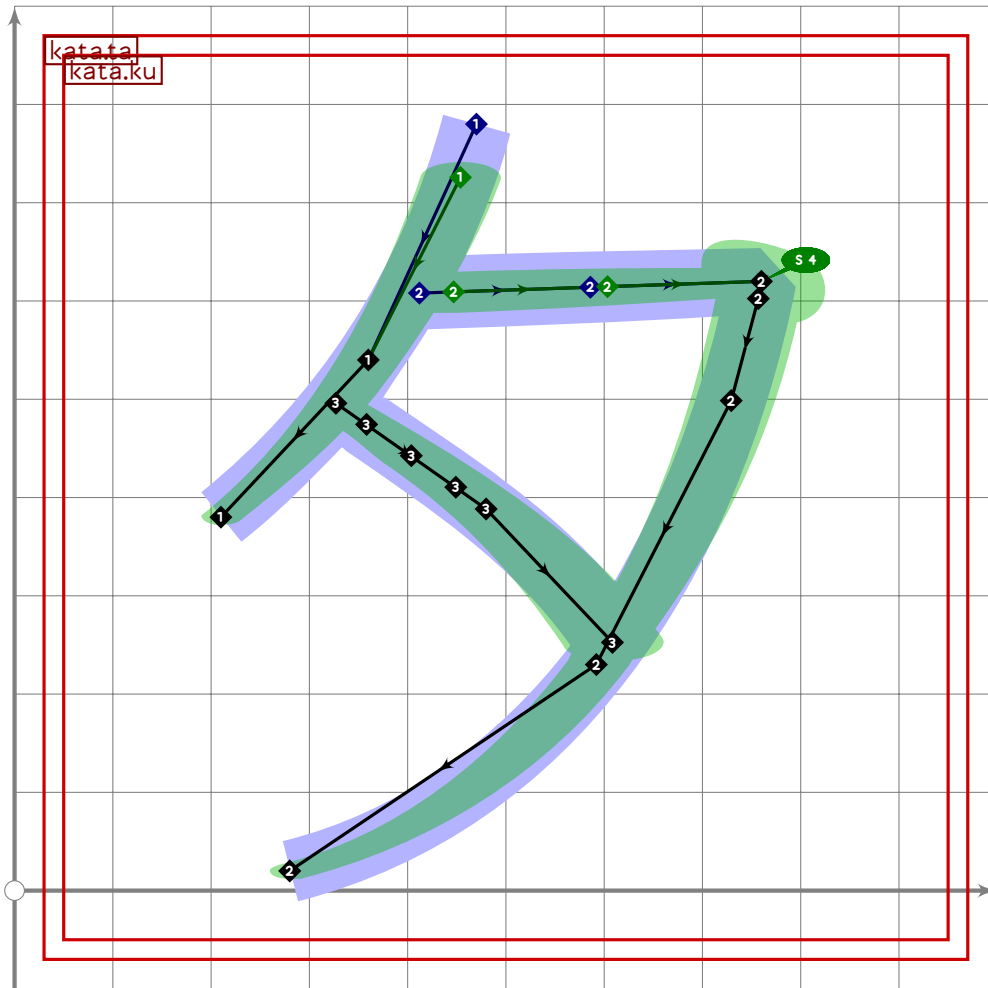
KATA

Katakana Tachitsuteto/Dajizudedo

```

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

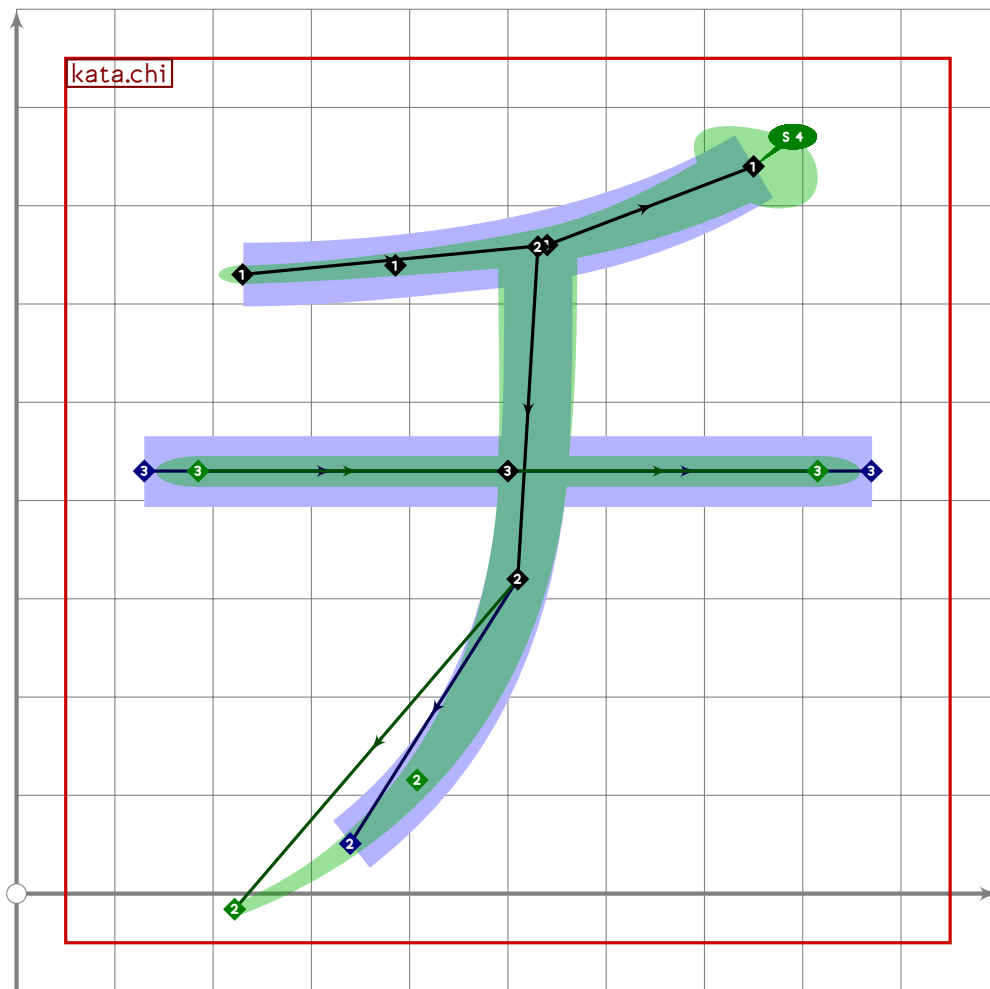
```



```

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)-(1.9,1.9));
283   expand_pbox;
284 enddef;

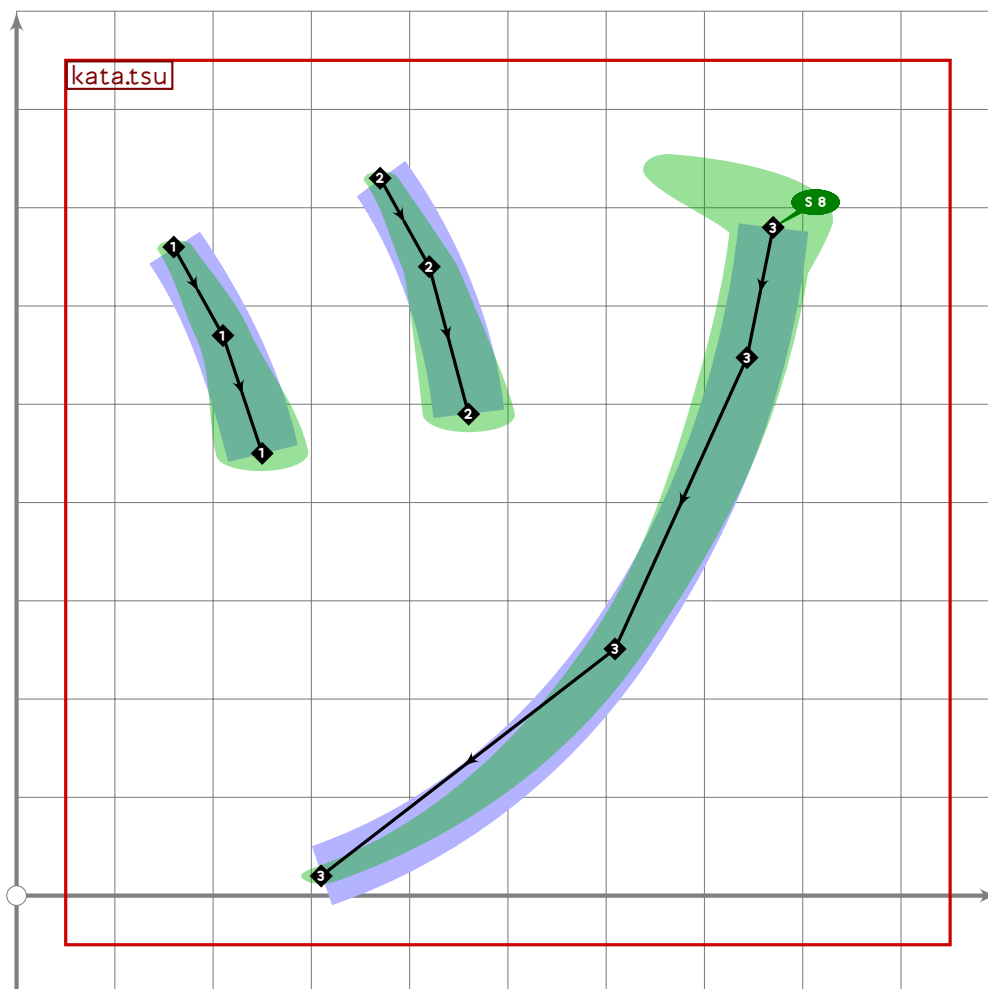
```



```

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));

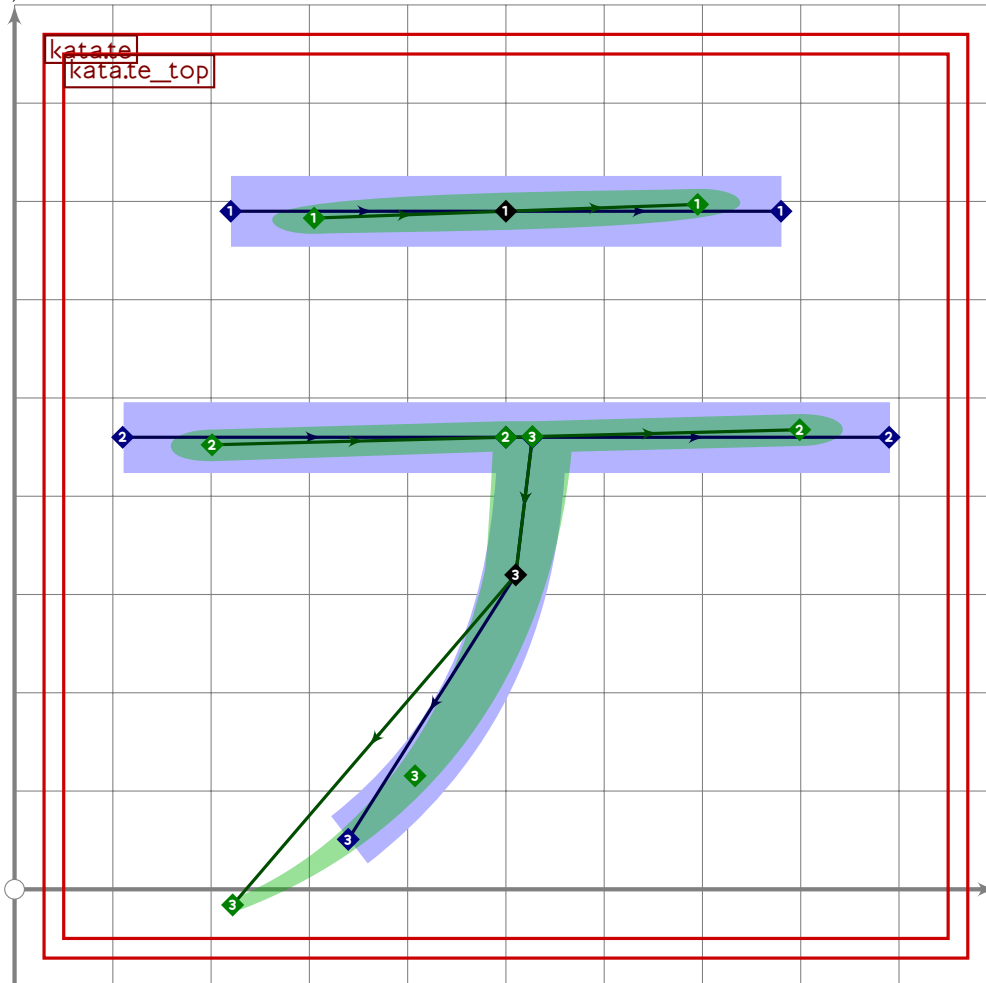
```


U+30C6
tsuku.uni30C6

```

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;

```

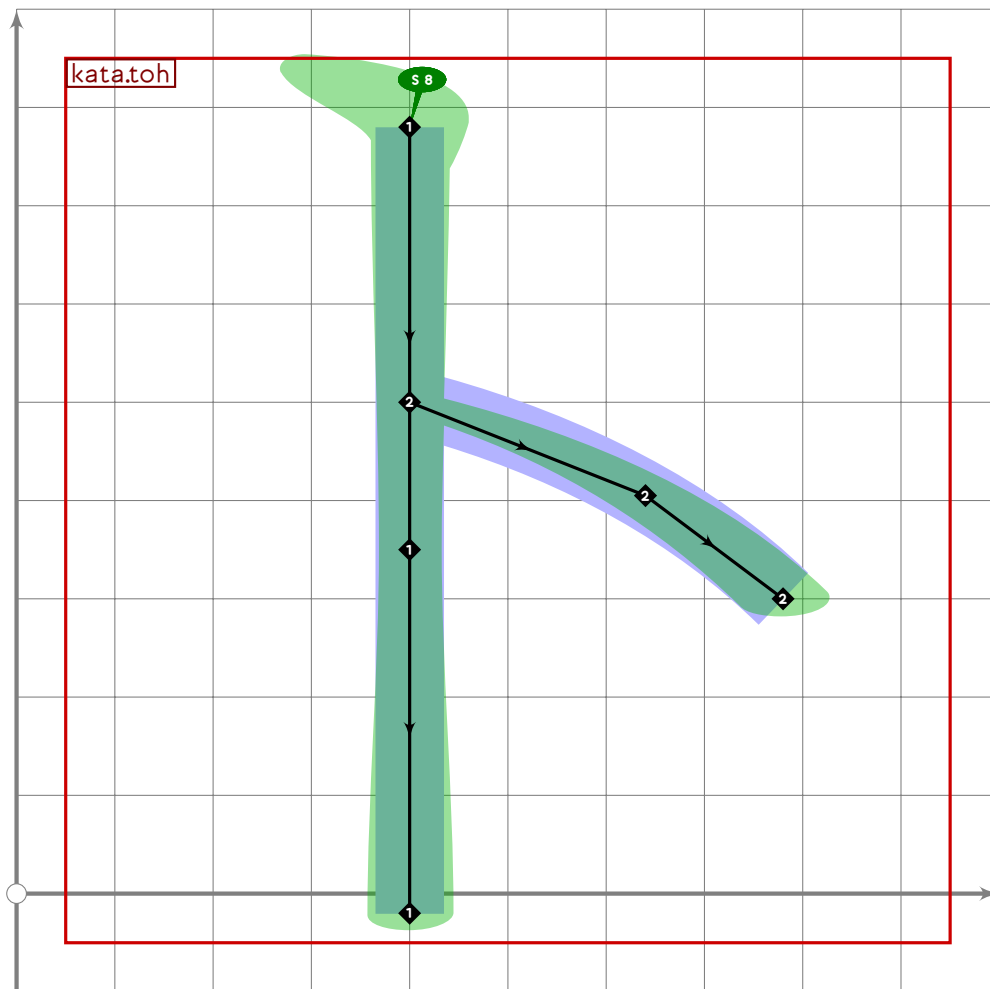


```

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;

```

KATA



```

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 1.1..(640,405)..(780,300),
355     (1.3,1.3)–(1.6,1.6)–(1.8,1.8));
356   expand_pbox;
357 enddef;
358

```

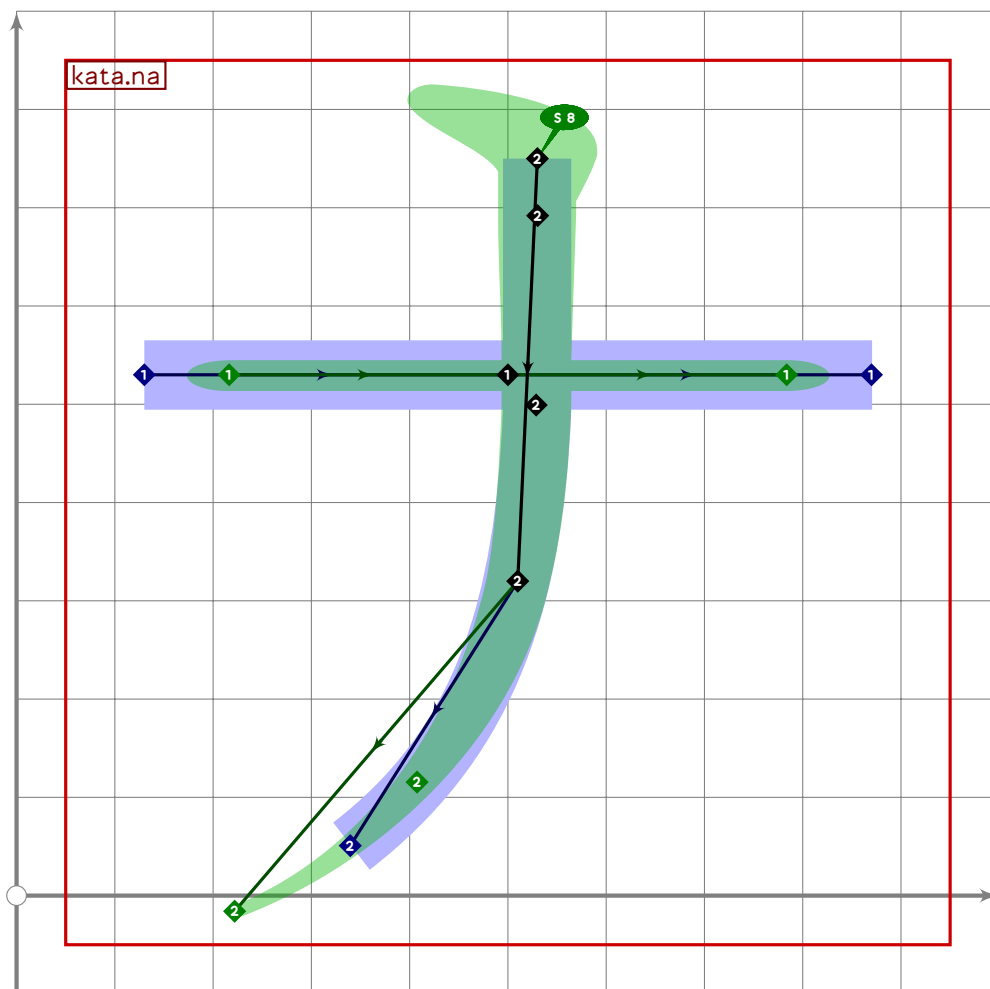
KATA

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;

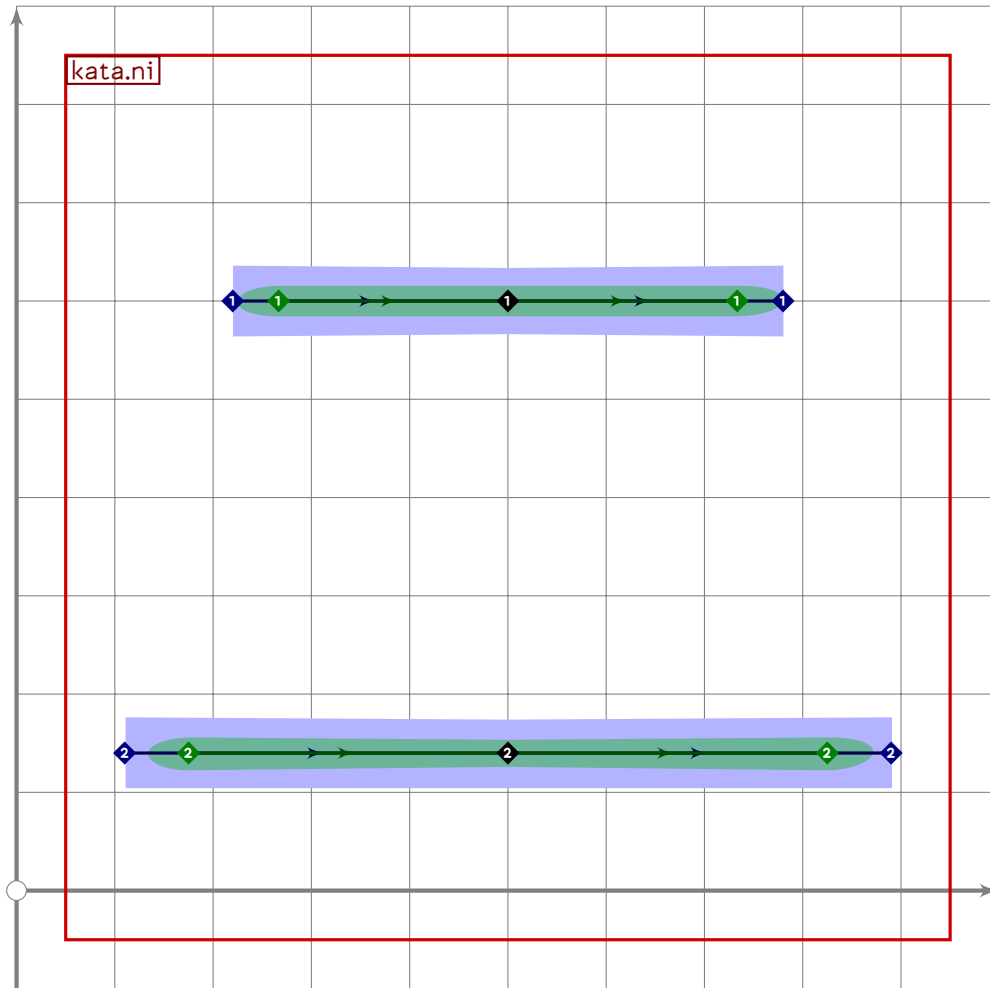
```



```

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;

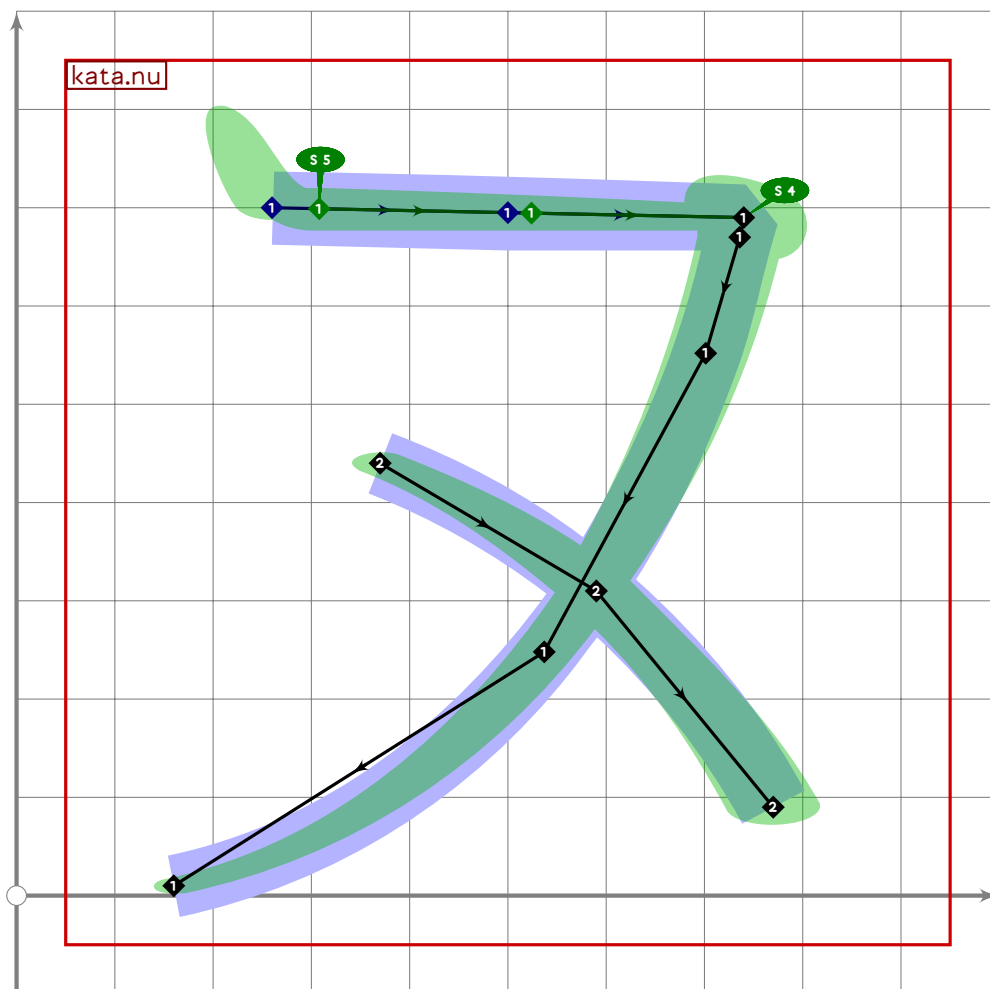
```



```

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;

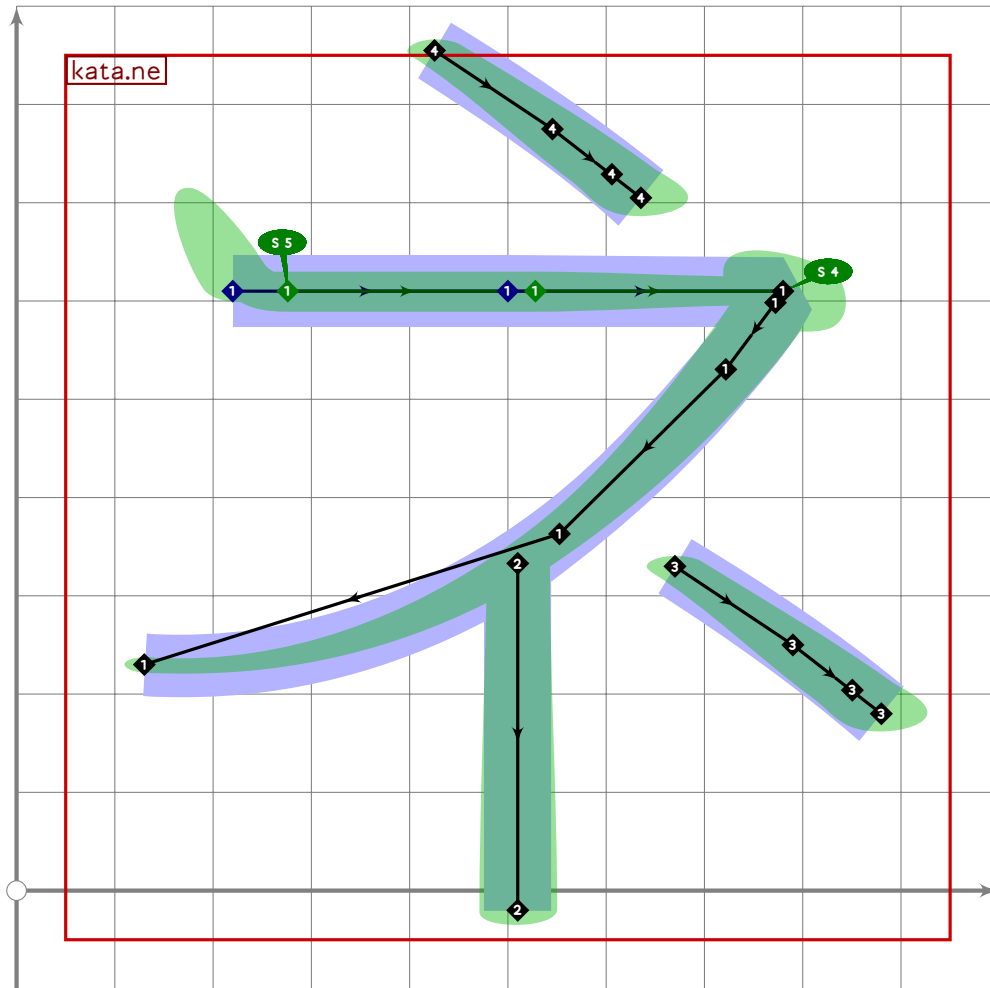
```



```

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;

```



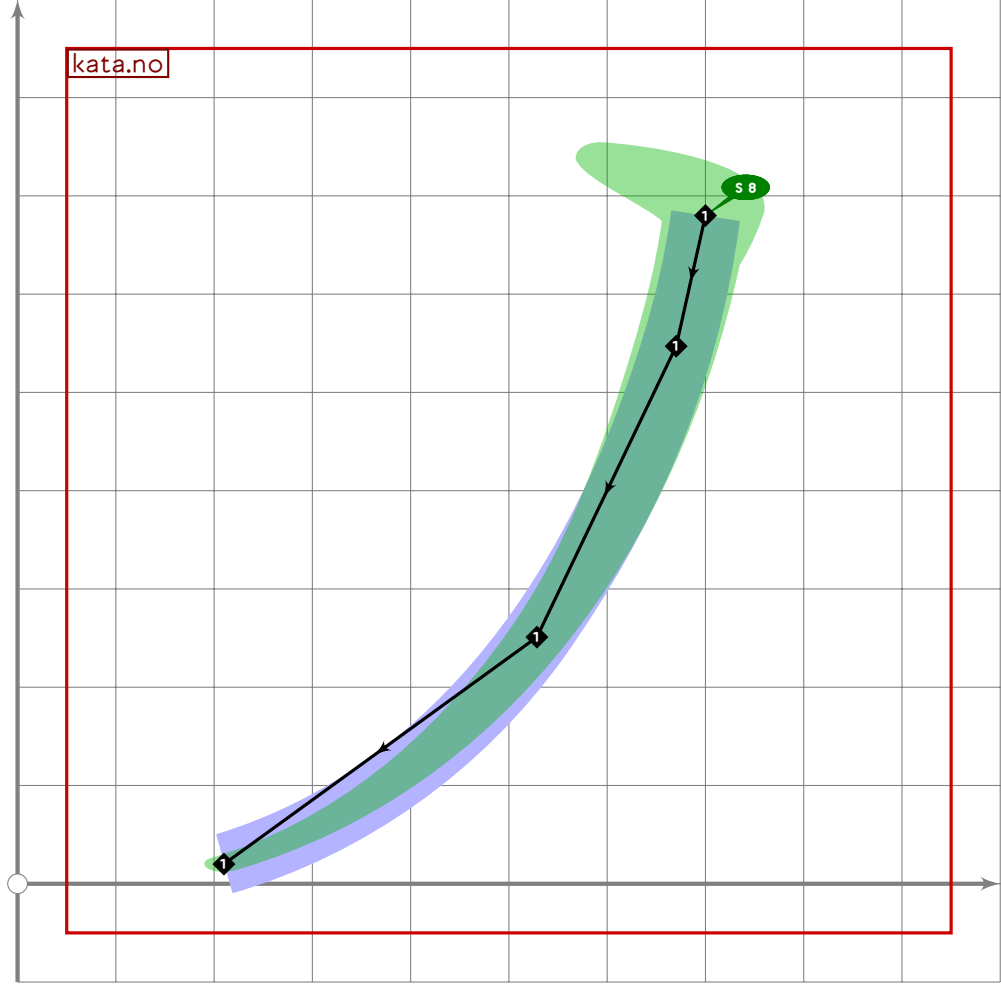
```

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),

```

U+30CE
tsuku.uni30CE

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

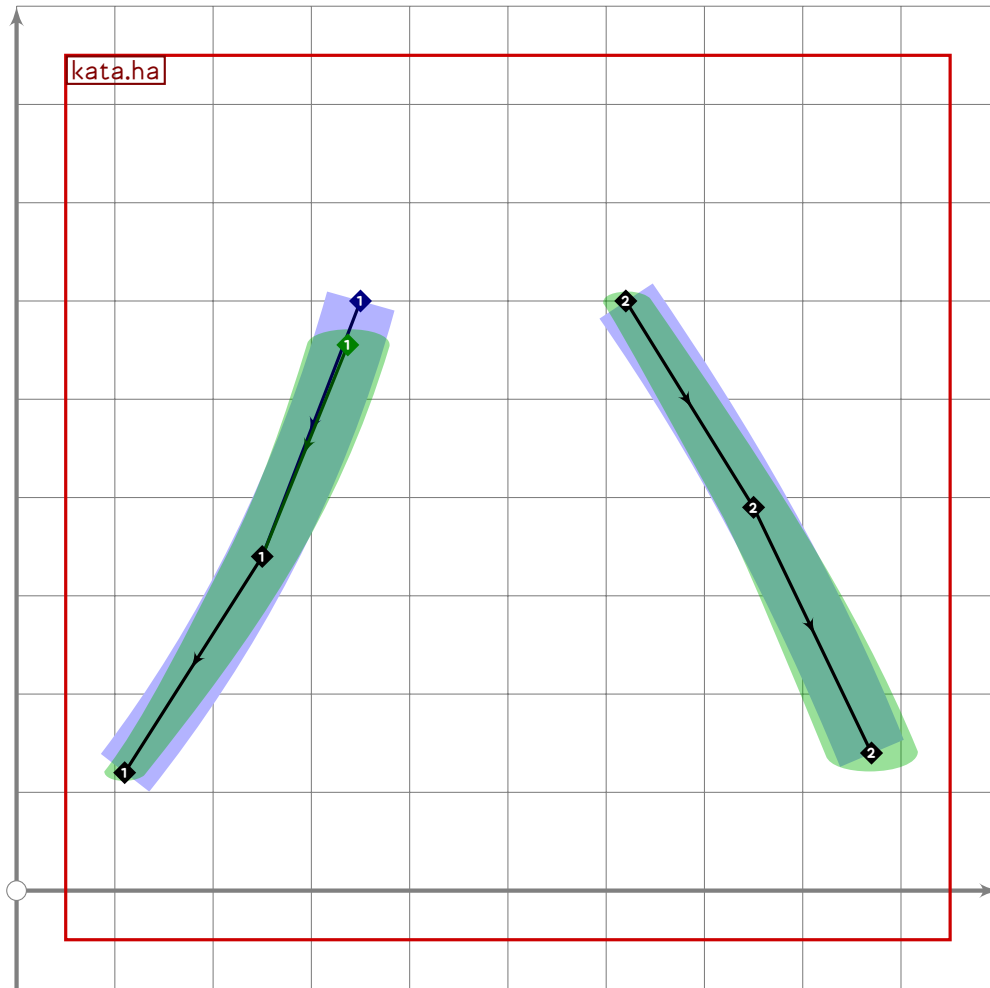


```
427  
428 vardef kata.no =  
429   push_pbox_toexpand("kata.no");  
430  
431   kata.no_stroke((700,680),(210,20));  
432   set_boserif(0,0,8);  
433   expand_pbox;  
434 enddef;  
435
```

KATA

Katakana Hahifuheho/Babibubebo/Papipupepo

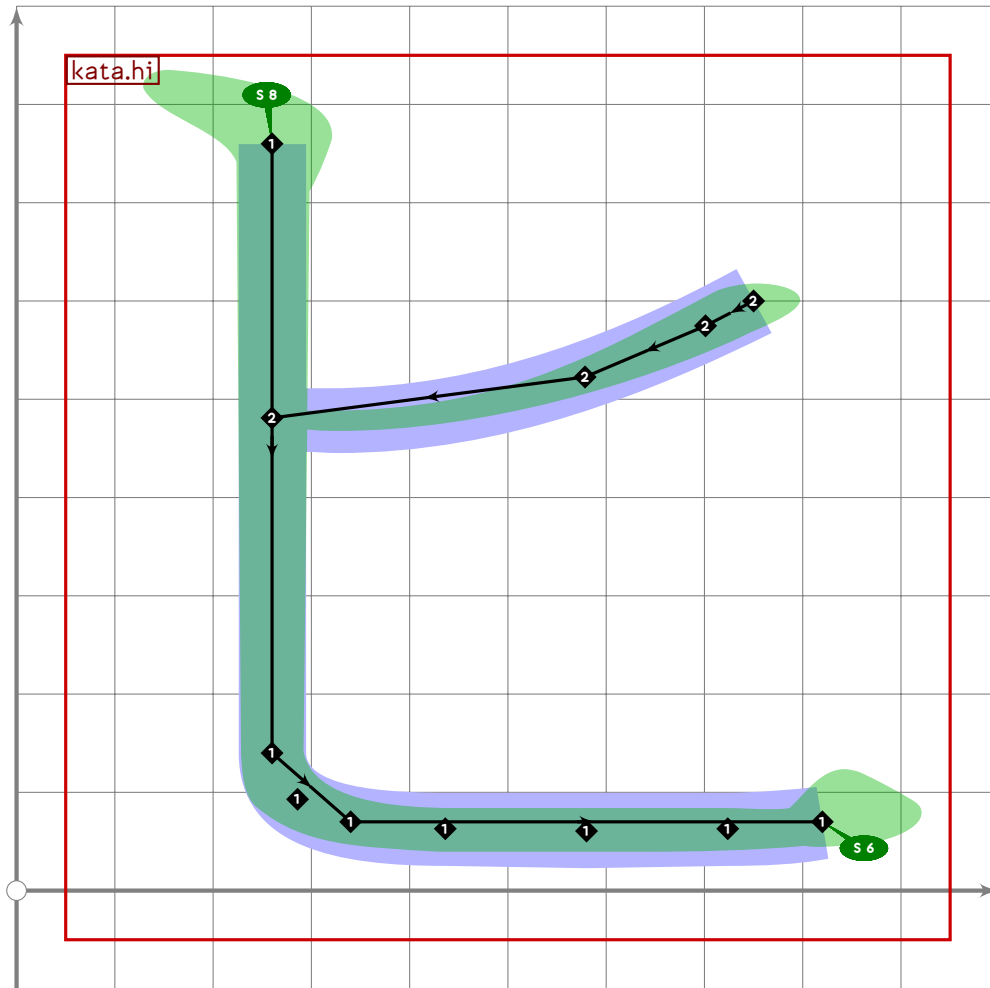
```
436 %%%%%%%%% KATAKANA HAHIFUHEHO/BABIBUBEBO/PAPIPUPEPO
```



```

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;

```

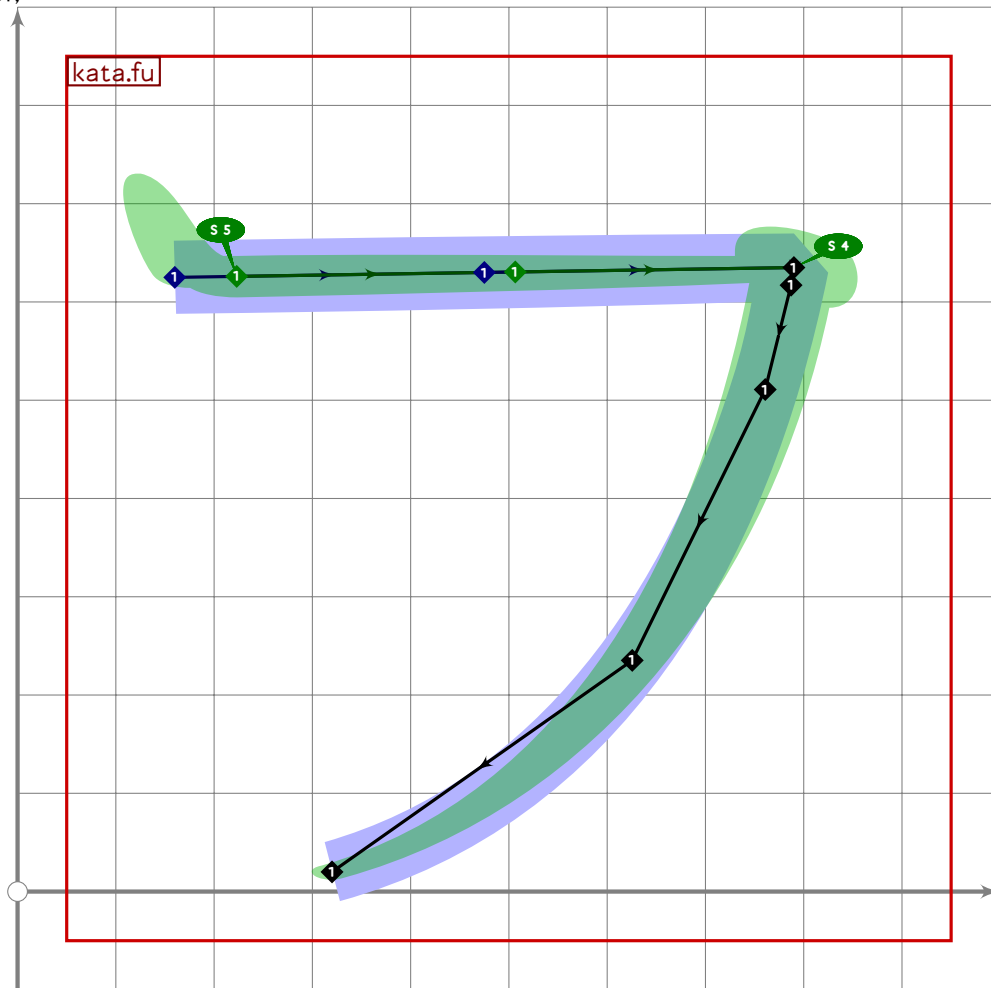



```

449
450 vardef kata.hi =
451   push_pbox_toexpand("kata.hi");
452
453   push_stroke((260,760)-(260,140){dir 274}..(340,70)..tension 21..(820,70),
454     (0.84,2.18)-(1.4,1.4)-(2.1,2.1)-(1.9,1.9));
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)-(1.9,1.9)-(1.5,1.5)-oldq);
467   set_botip(0,2,0);
468   set_boserif(0,0,5);
469   set_boserif(0,2,4);

```

470 enddef;



471

472 vardef kata.fu =

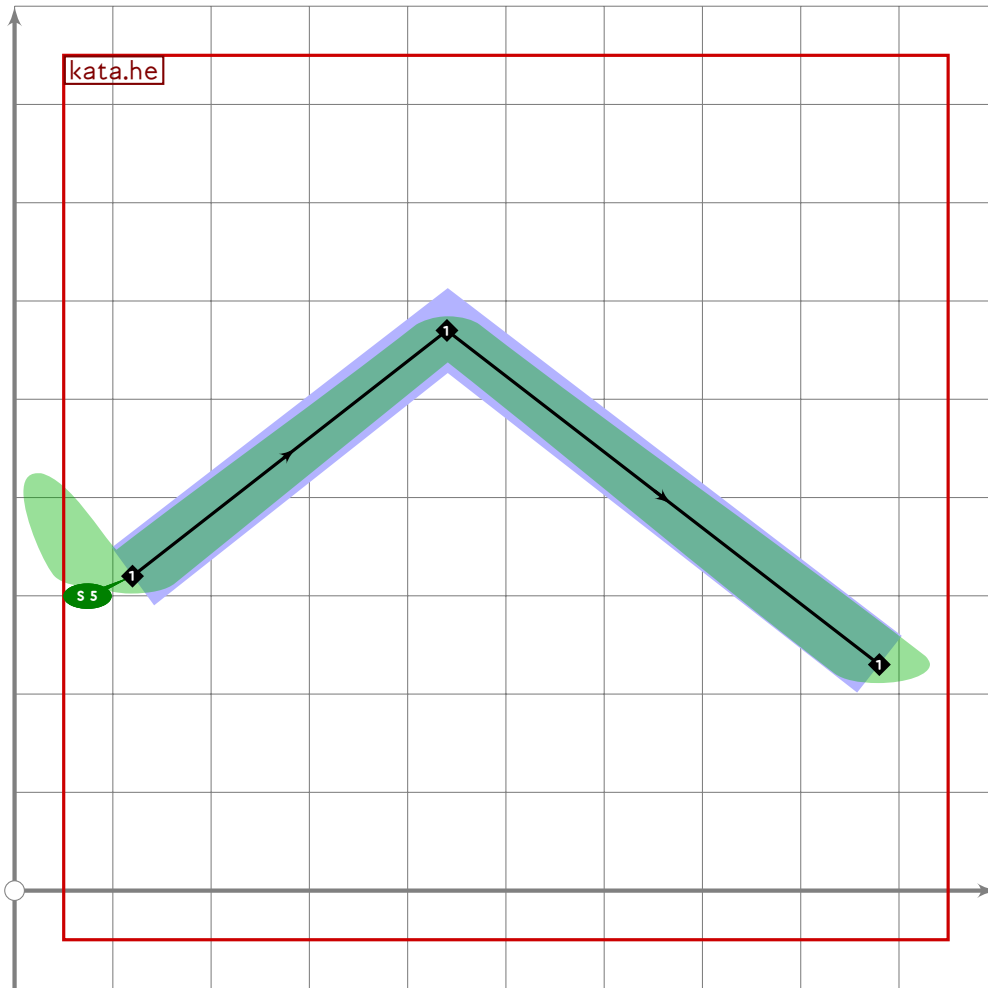
473 push_pbox_toexpand("kata.fu");

474

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

476 expand_pbox;

477 enddef;



```

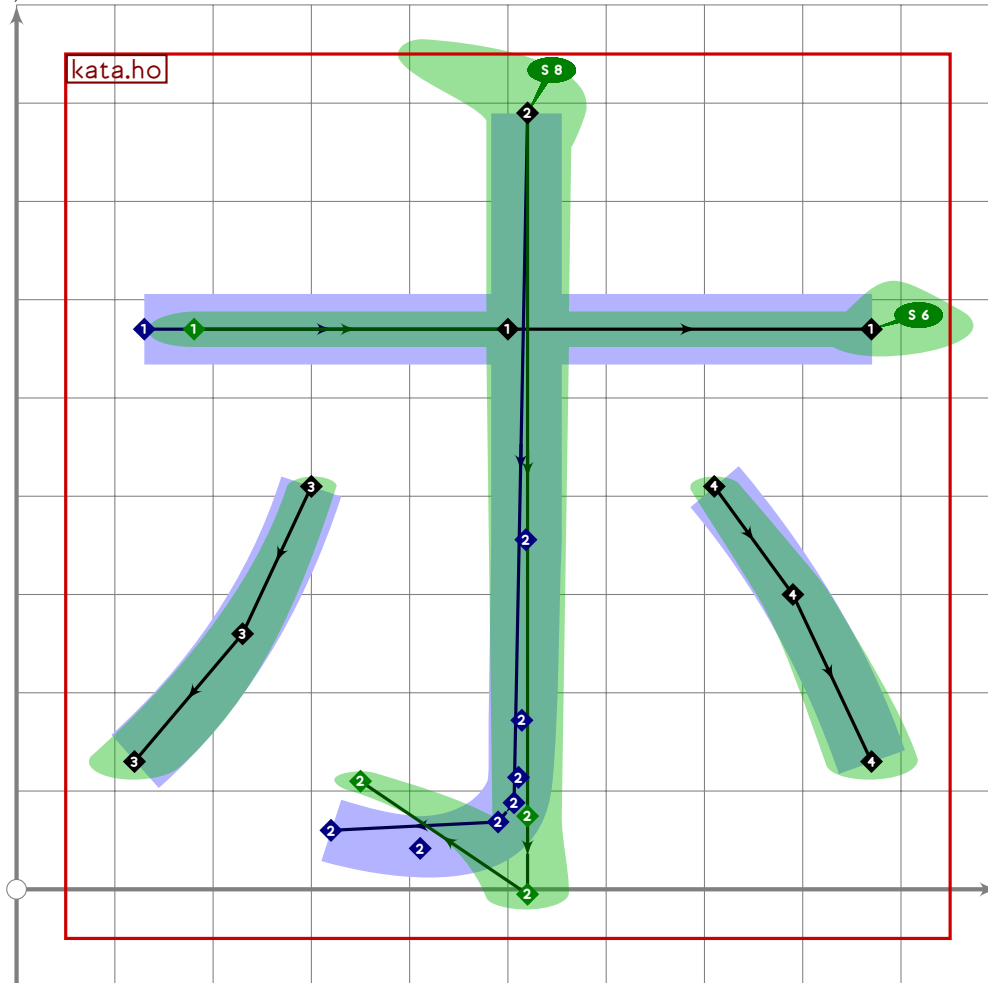
478
479 vardef kata.he =
480   push_pbox_toexpand("kata.he");
481
482   push_stroke((120,320)–(440,570)–(880,230),
483     (1.8,1.8)–(1.5,1.5)–(1.9,1.9));
484   set_botip(0,1);
485   set_boserif(0,0,5);
486   expand_pbox;
487 enddef;
488
489 vardef kata.ho_centre(expr pta,ptb) =
490   push_stroke(begingroup
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;

```

```

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

```

U+30DE
tsuku.uni30DE

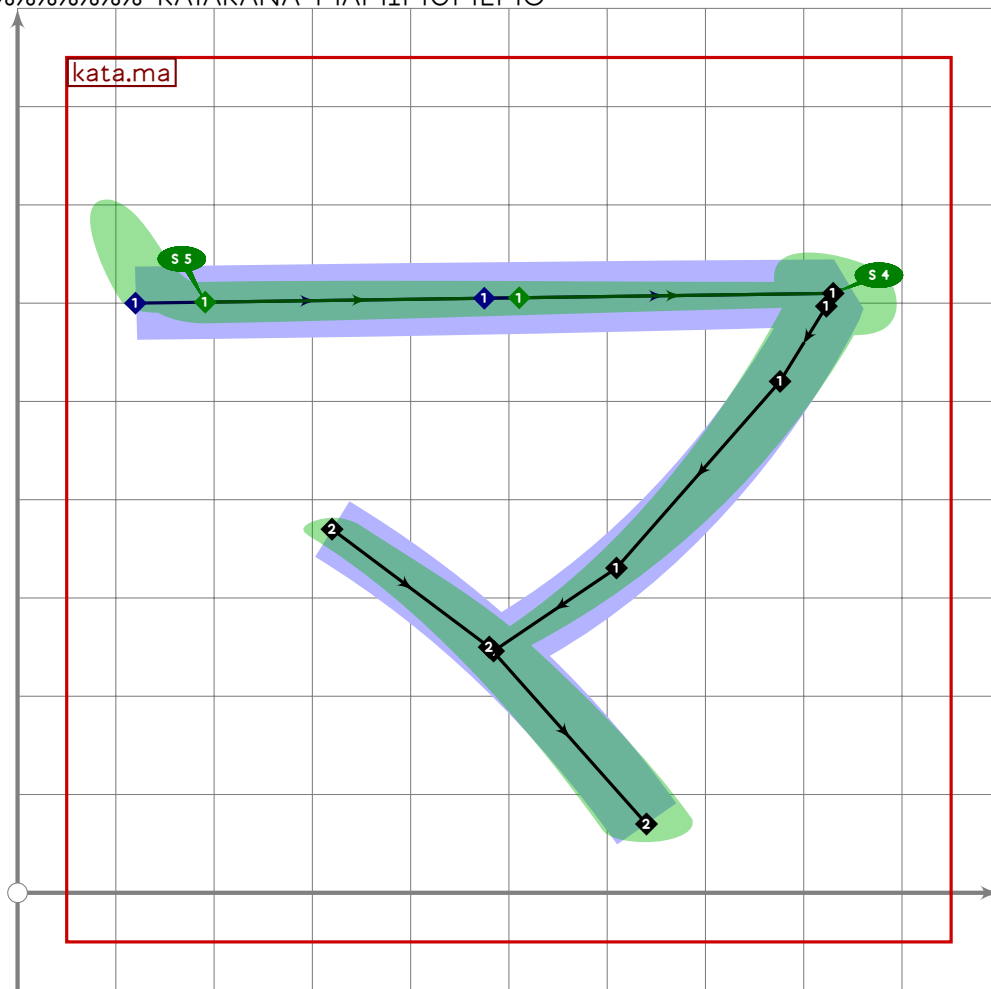
```

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



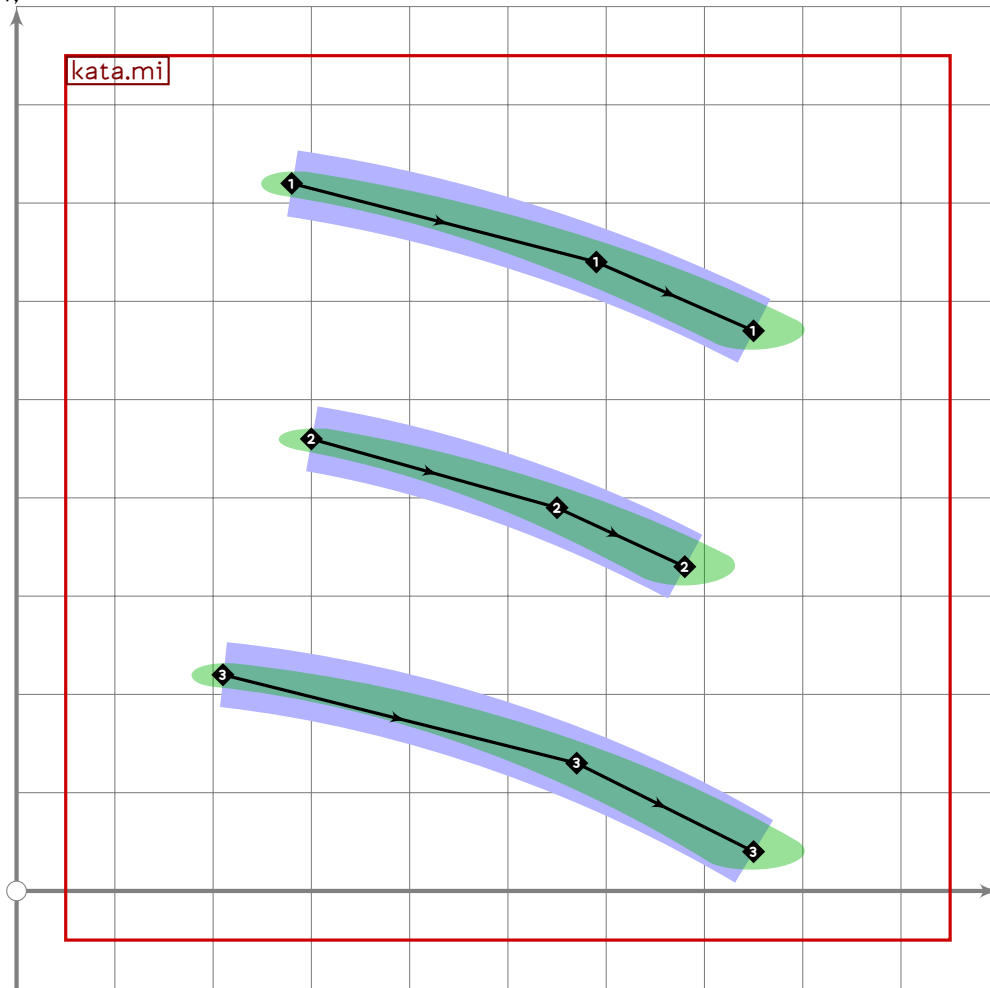
KATA

```

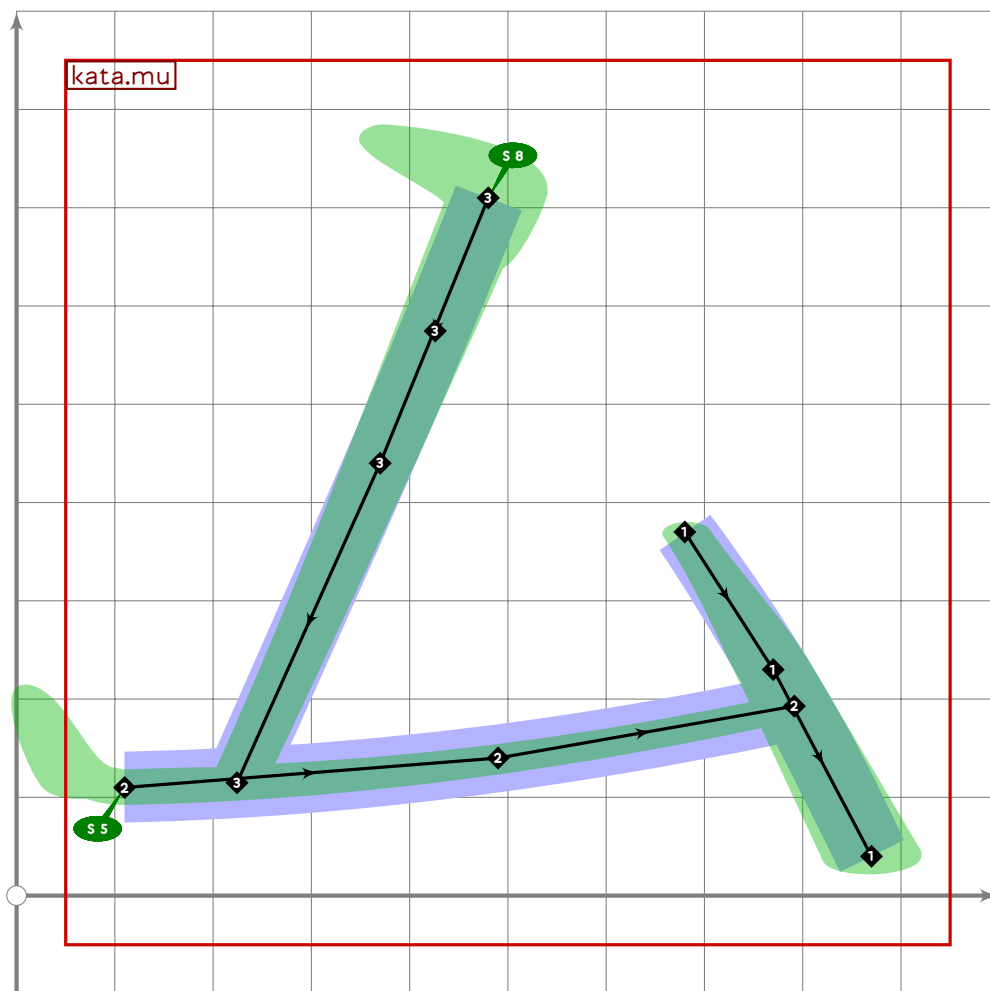
529
530 vardef kata.ma =
531   push_pbox_toexpand("kata.ma");
532
533   kata.fu_stroke((120,600),(830,610),(200,180));
534
535   push_stroke((320,370)..(480,250)..(640,70),
536     (1.3,1.3)-(1.6,1.6)-(1.8,1.8));
537   replace_stroke(-1)(subpath (0,xpart (oldp intersectiontimes

```

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



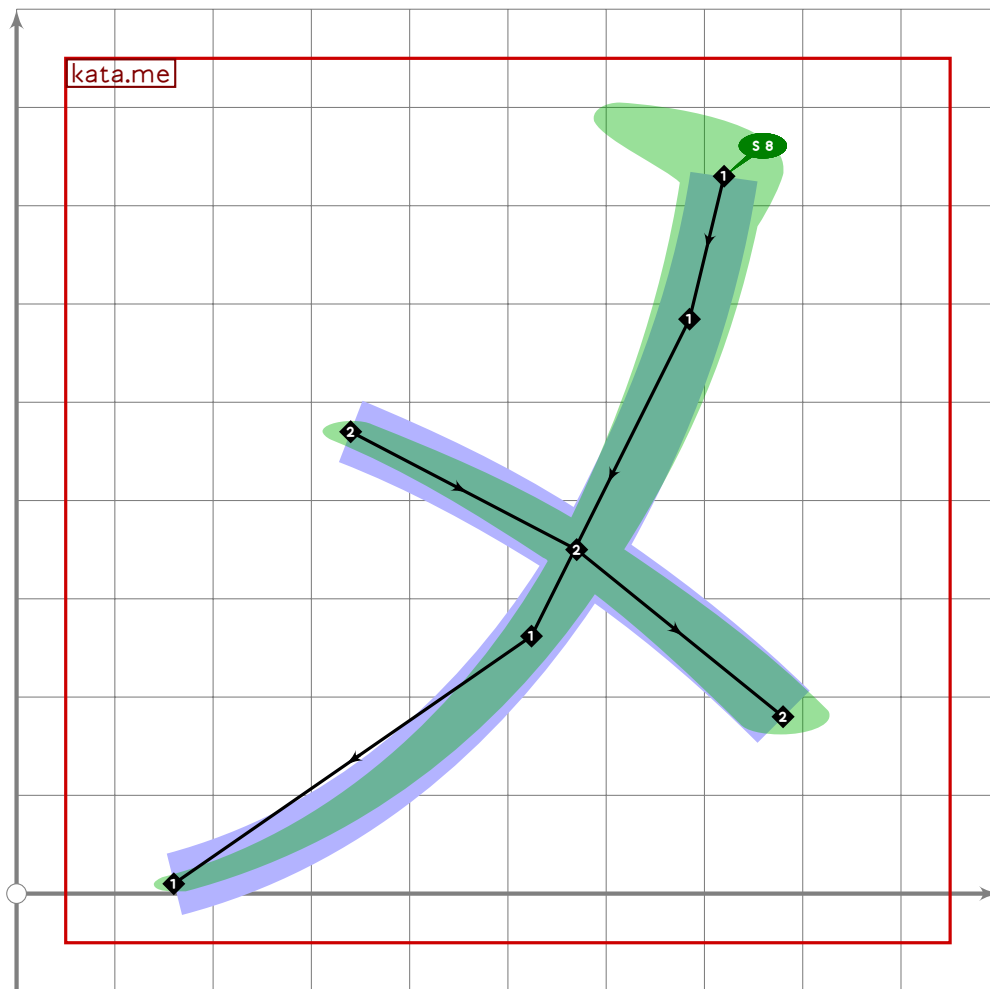
```
541
542 vardef kata.mi =
543   push_pbox_toexpand("kata.mi");
544
545   push_stroke((280,720)..(590,640)..(750,570),
546     (1.4,1.4)-(1.7,1.7)-(1.9,1.9));
547
548   push_stroke((300,460)..(550,390)..(680,330),
549     (1.4,1.4)-(1.7,1.7)-(1.9,1.9));
550
551   push_stroke((210,220)..(570,130)..(750,40),
552     (1.4,1.4)-(1.7,1.7)-(1.9,1.9));
553   expand_pbox;
554 enddef;
```



```

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_strokep(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_strokep(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;

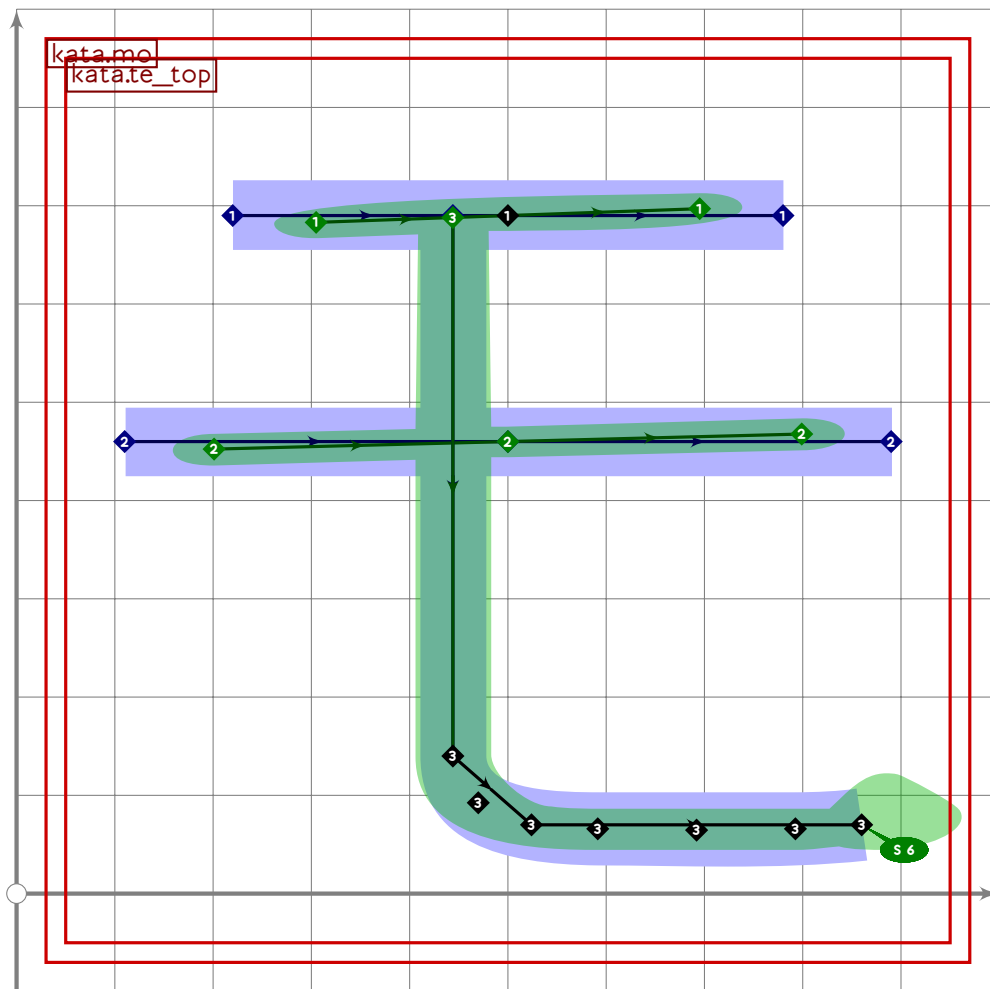
```



```

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 2.1..(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

```

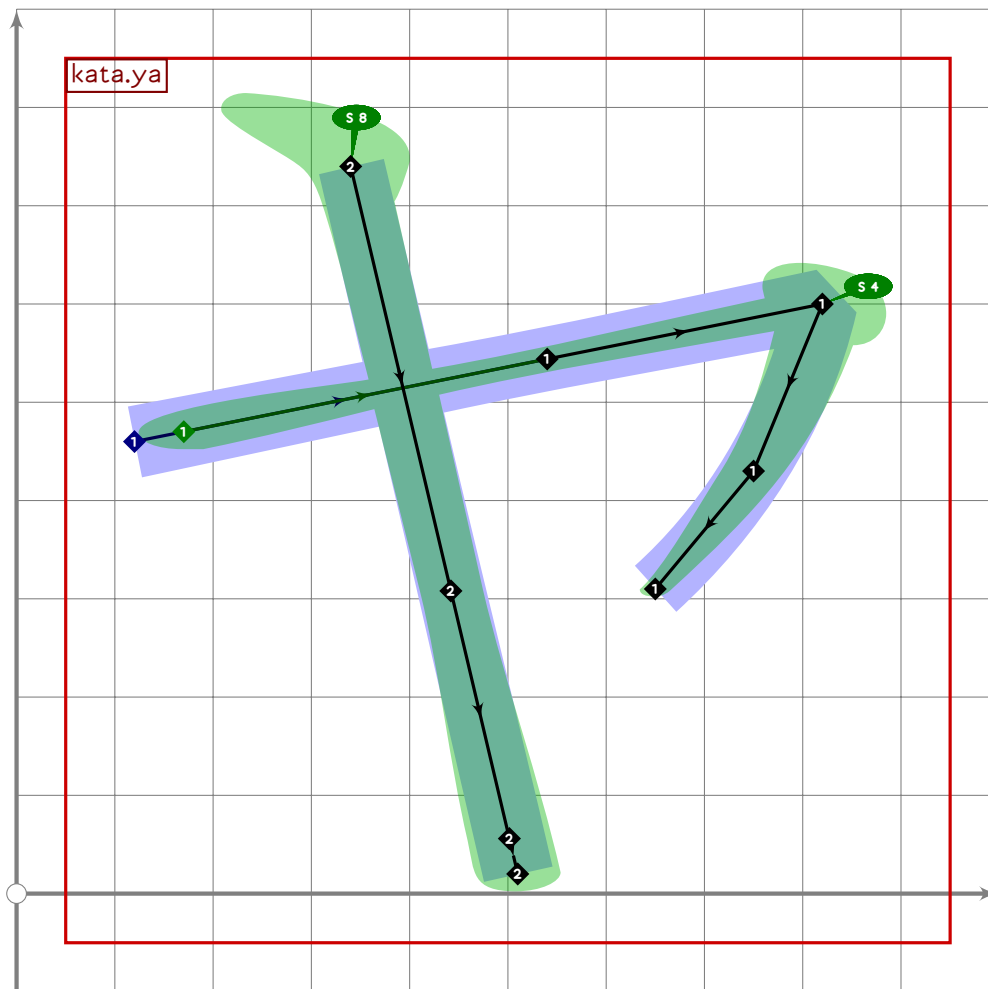
KATA

Katakana Yayuyo

```

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

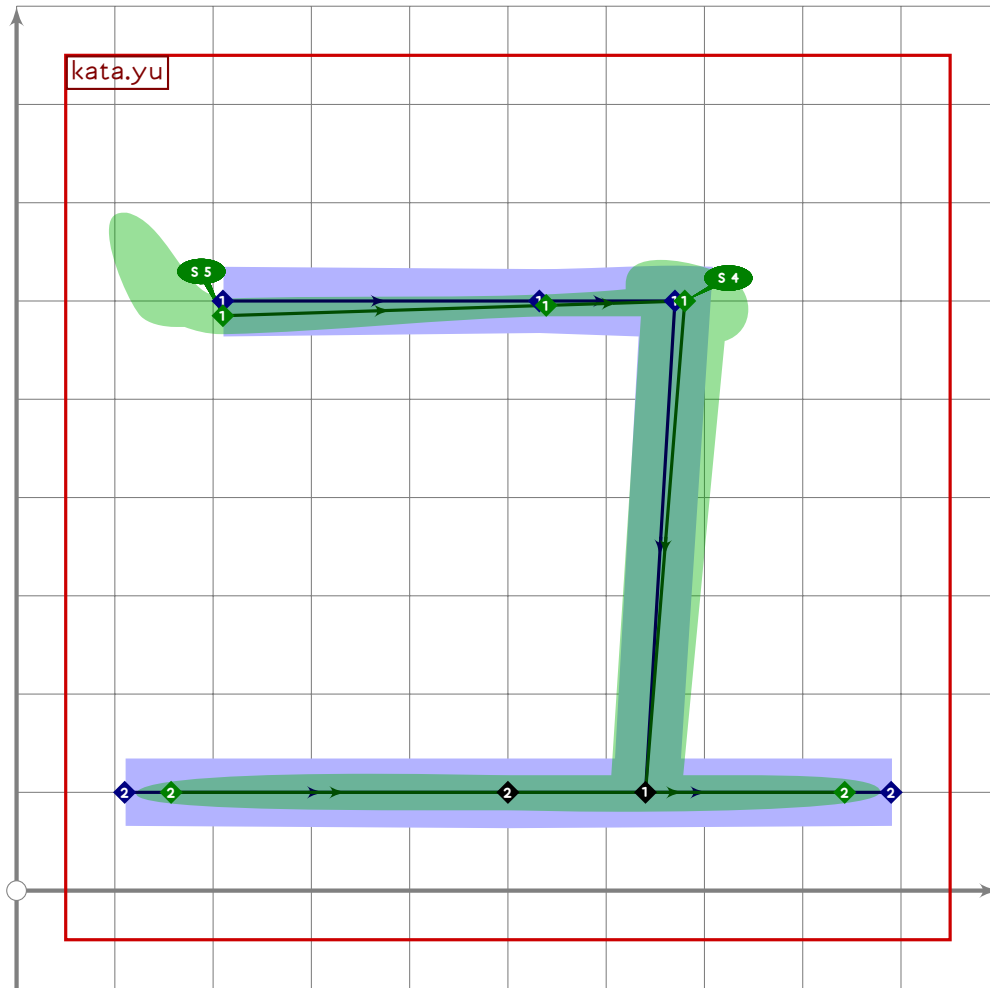
```



```

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;

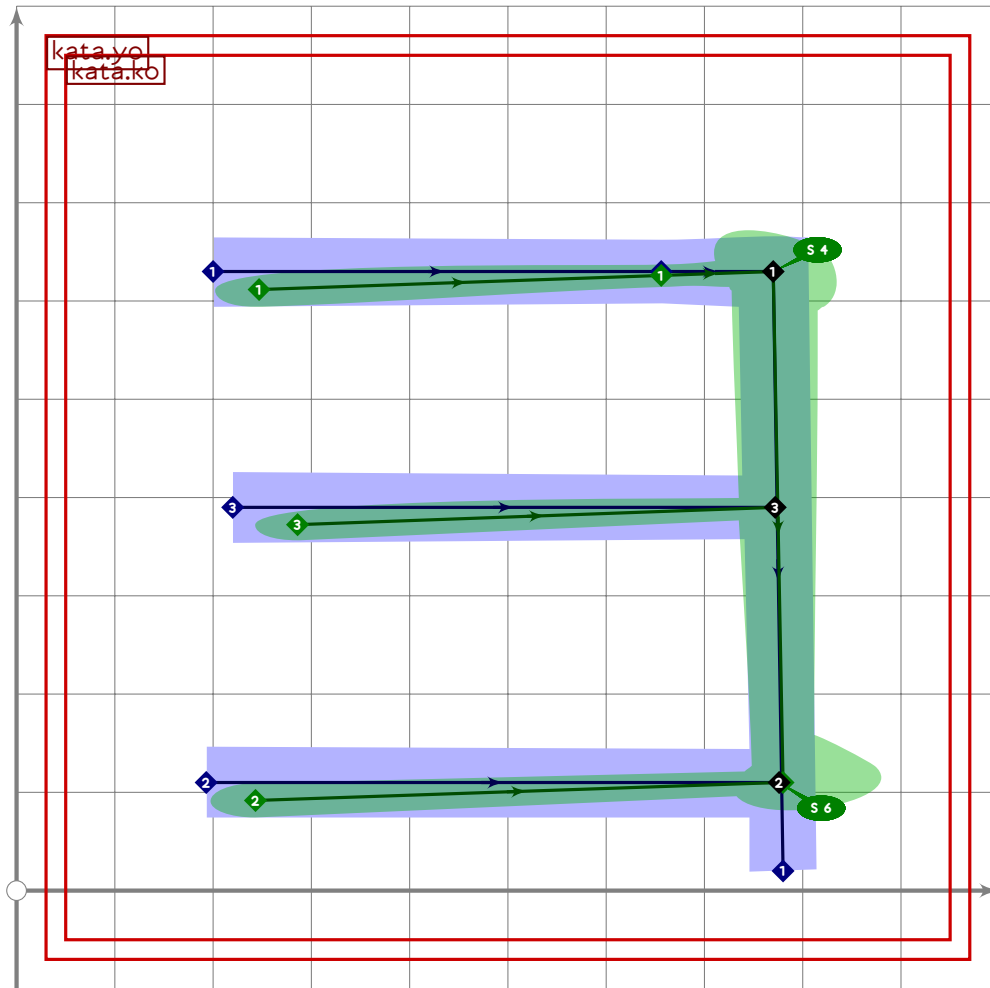
```



```

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   set_boserif(0,0,5);
628   set_boserif(0,2,6);
629   expand_pbox;
630 enddef;

```



```

631
632 vardef kata.yo =
633   push_pbox_toexpand("kata.yo");
634
635   kata.ko;
636
637   push_stroke((220,390-20*mincho)-(772,390),
638     (0.77,2.7)-(1.3,1.3));
639   set_boserif(0,0,5);
640   expand_pbox;
641 enddef;
642

```

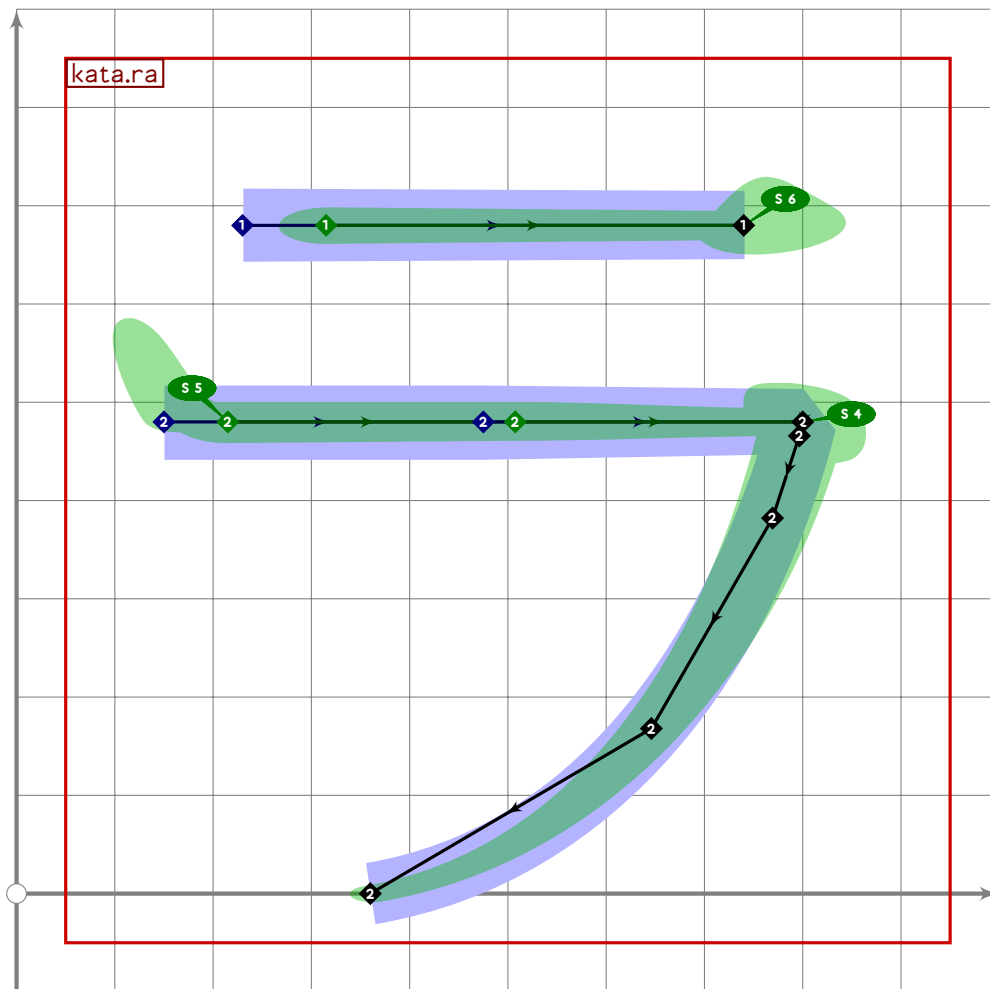
KATA

Katakana Rarirurero

```

643 %%%%%%%%% KATAKANA RARIRURERO

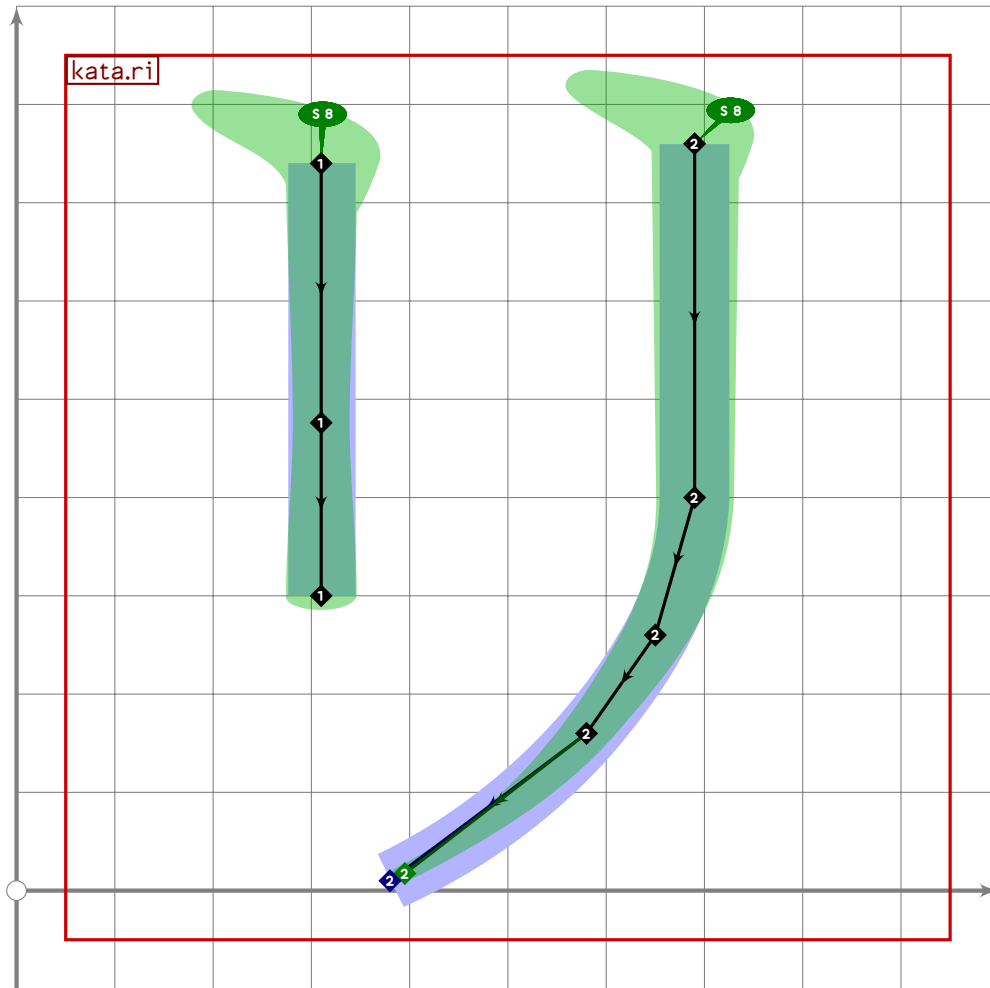
```



```

644
645 vardef kata.ra =
646   push_pbox_toexpand("kata.ra");
647
648   push_stroke((230,680)-(740,680),
649     (0.68,3.12)-(1.6,1.6));
650   set_boserif(0,0,5);
651   set_boserif(0,1,6);
652
653   kata.fu_stroke((150,480),(800,480),(360,0));
654   expand_pbox;
655 enddef;

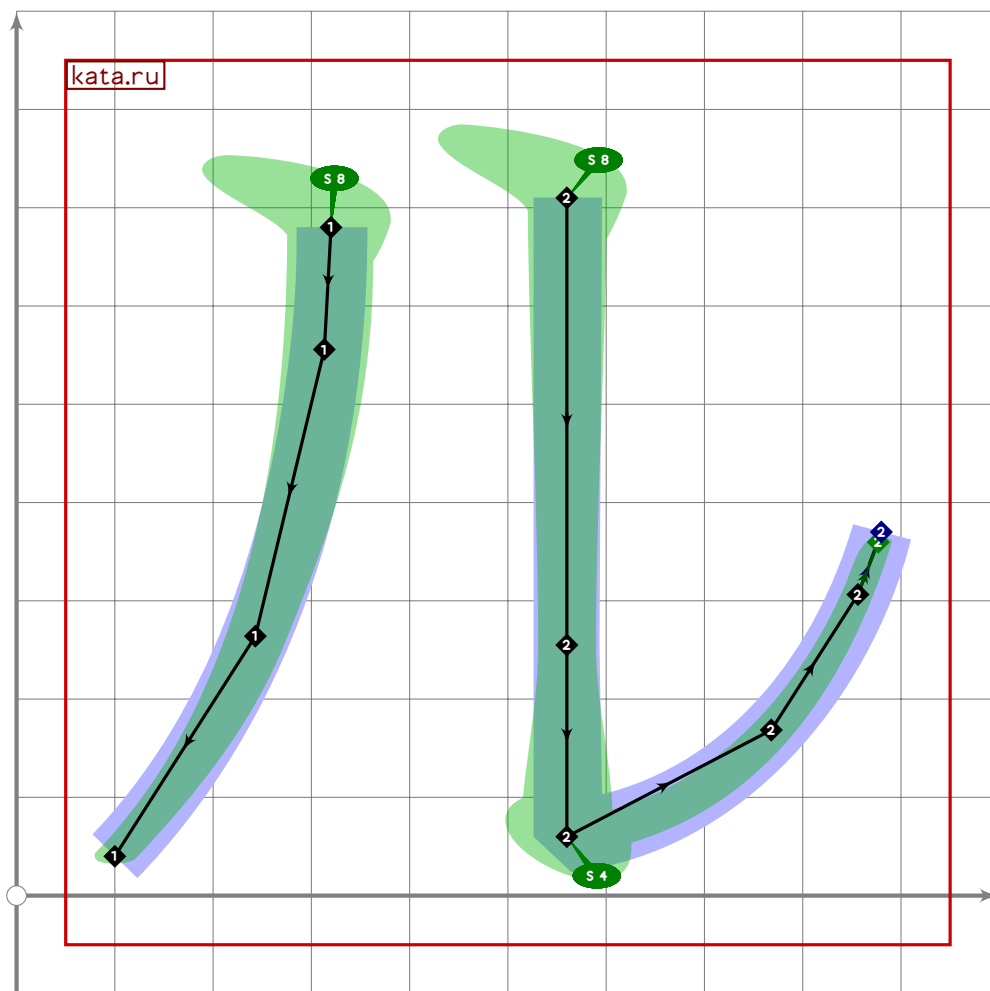
```



```

656
657 vardef kata.ri =
658   push_pbox_toexpand("kata.ri");
659
660   push_stroke((310,740)–(310,300),
661     (1.5,1.5)–(1.3,1.3)–(1.5,1.5));
662   replace_strokep(0)(insert_nodes(oldp)(0.6));
663   set_boserif(0,0,8);
664
665   push_stroke((690,760)–(690,400){dir 267}..(650,260)..(580,160)..(380,10),
666     (1.7,1.7)–(1.6,1.6)–(1.5,1.5)–(1.3,1.3)–(0.8,1));
667   set_boserif(0,0,8);
668   expand_pbox;
669 enddef;

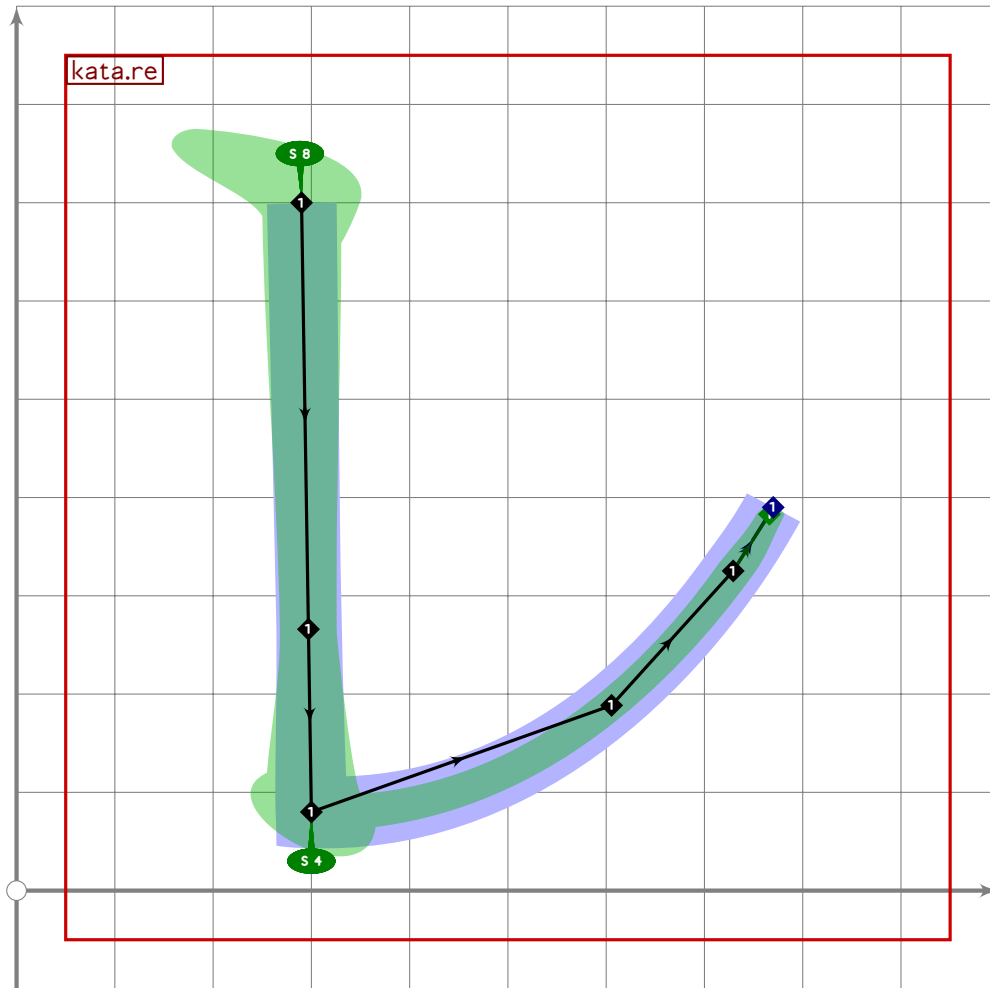
```



```

670
671 vardef kata.ru =
672   push_pbox_toexpand("kata.ru");
673
674   kata.no_stroke((320,680),(100,40));
675   set_boserif(0,0,8);
676
677   kata.no_stroke((880,370),(560,60));
678   replace_strokep(0)(insert_nodes((560,710)-reverse oldp)(0.7));
679   replace_strokeq(0)((1.6,1.6)-(1.3,1.3)-(1.8,1.8)-
680     (1.2,1.2)-(1,1)-(0.8,1));
681   set_botip(0,2,0);
682   set_boserif(0,0,8);
683   set_boserif(0,2,4);
684   expand_pbox;
685 enddef;

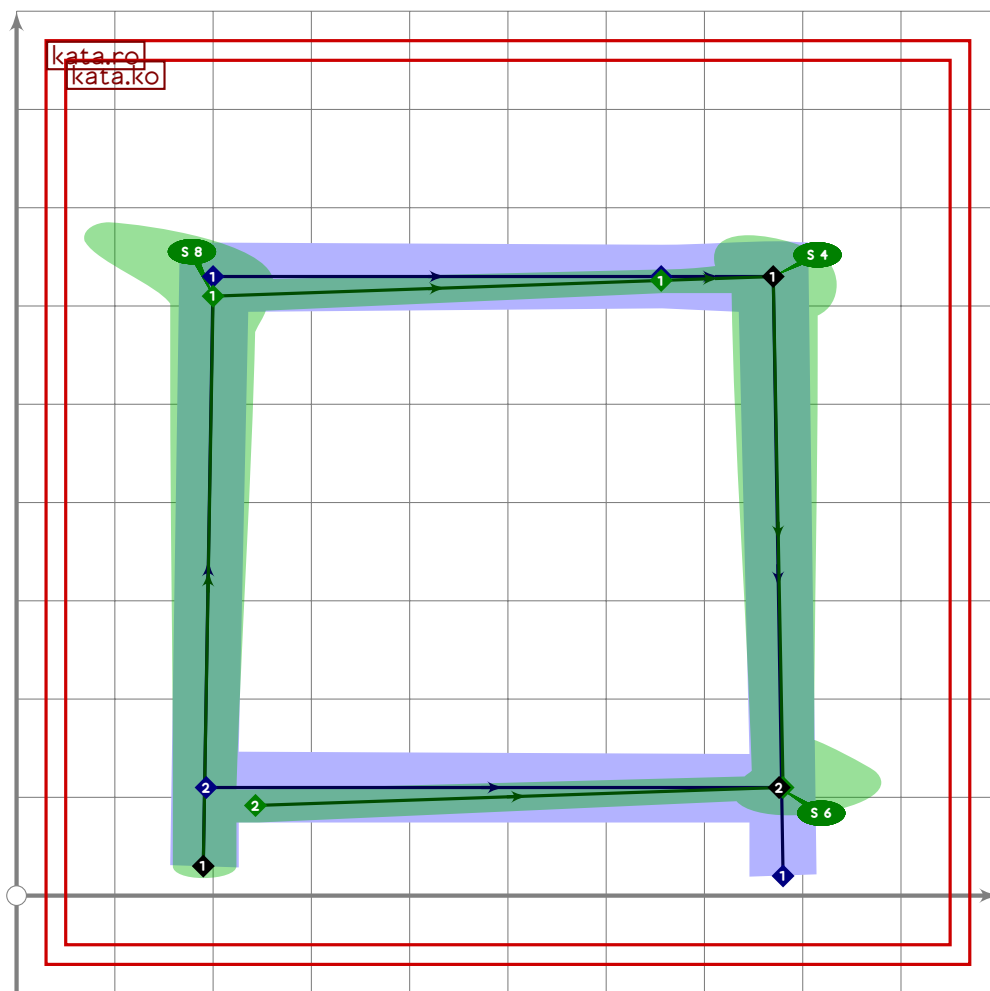
```



```

686
687 vardef kata.re =
688   push_pbox_toexpand("kata.re");
689
690   kata.no_stroke((770,390),(300,80));
691   replace_strokep(0)(insert_nodes((290,700)-reverse oldp)(0.7));
692   replace_strokeq(0)((1.6,1.6)-(1.3,1.3)-(1.8,1.8)-
693     (1.2,1.2)-(1.1,1.1)-(0.8,1));
694   set_botip(0,2,1);
695   set_boserif(0,0,8);
696   set_boserif(0,2,4);
697   expand_pbox;
698 enddef;

```

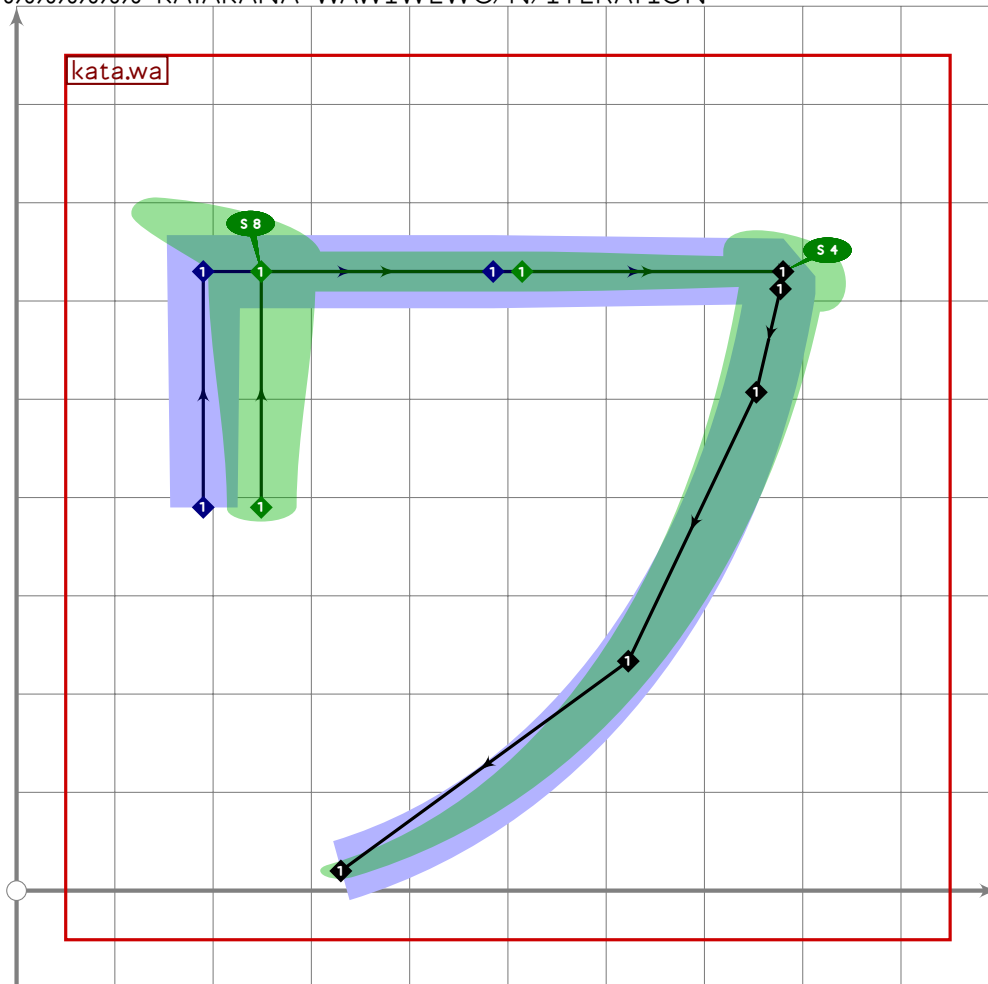
```

699
700 vardef kata.ro =
701   push_pbox_toexpand("kata.ro");
702
703   kata.ko;
704
705   replace_strokep(-1)((190,30)-oldp);
706   replace_strokeq(-1)((1,4,1,4)-(1,7,1,7)-(subpath (1,infinity) of oldq));
707   set_botip(-1,1,1);
708   set_botip(-1,2,whatever);
709   set_botip(-1,3,1);
710   set_boserif(-1,0,whatever);
711   set_boserif(-1,1,8);
712   set_boserif(-1,2,whatever);
713   set_boserif(-1,3,4);
714   set_boserif(0,0,whatever);
715   expand_pbox;
716 enddef;
717

```

Katakana Wawiwewo/N/Iteration

718 %%%%%%%%% KATAKANA WAWIWEWO/N/ITERATION

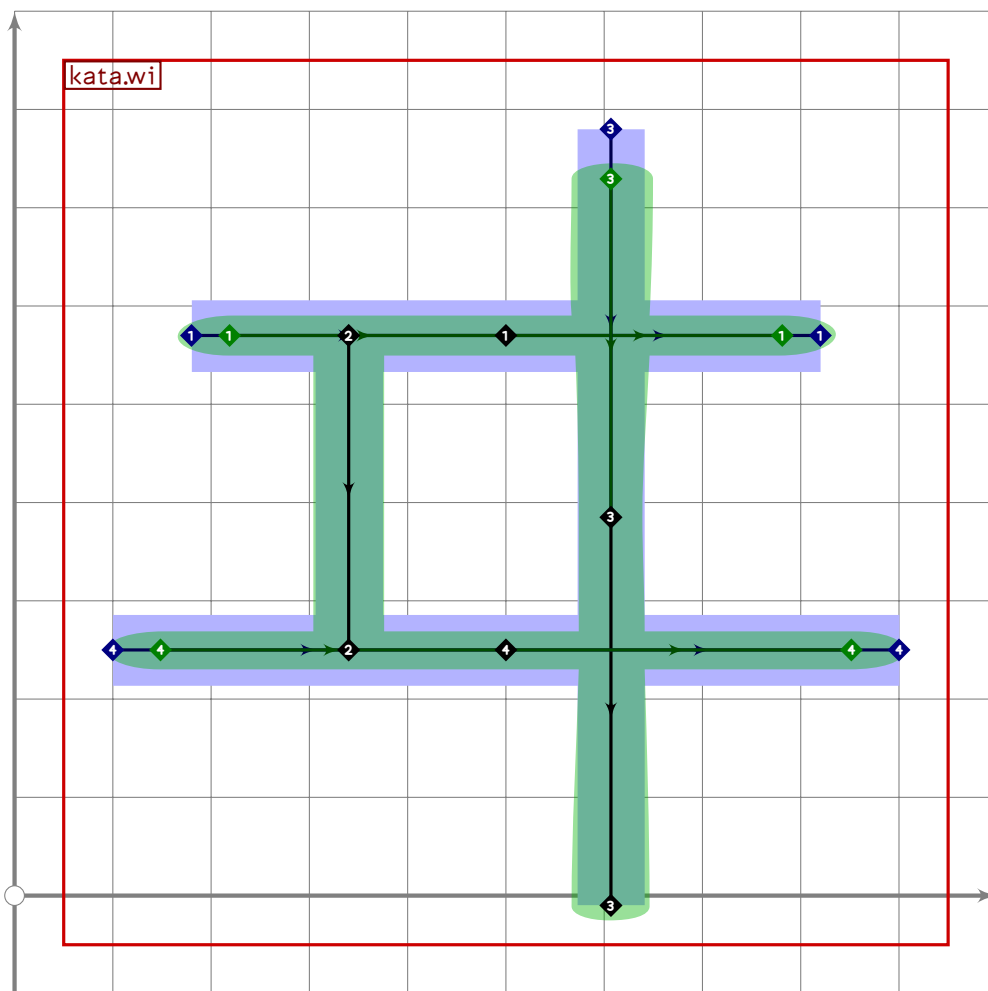


```

719
720 vardef kata.wa =
721   push_pbox_toexpand("kata.wa");
722
723   kata.fu_stroke((190,630),(780,630),(330,20));
724   replace_strokep(0)((xpart point 0 of oldp,390)-oldp);
725   replace_strokeq(0)((1.5,1.5)-oldq);
726   set_botip(0,1,1);
727   set_botip(0,3,0);
728   set_boserif(0,0,whatever);
729   set_boserif(0,1,8);
730   set_boserif(0,2,whatever);
731   set_boserif(0,3,4);
732   expand_pbox;
733 enddef;

```

U+30F0
tsuku.uni30F0



```

734
735 vardef kata.wi =
736   push_pbox_toexpand("kata.wi");
737
738   x1=100;
739   x2=180;
740   x3=0.25[x2,x5];
741   x4=0.667[x2,x5];
742   (x5+x2)/2=(x1+x6)/2=500;
743   y1=-10;
744   y2=250;
745   y3=570;
746   y4=780;
747   push_stroke((x2,y3)-(x5,y3),
748     (0.7,3.3)-(1.8,1.8)-(0.7,3.3));
749   replace_strokep(0)(insert_nodes(oldp)(0.5));
750   set_boserif(0,0,5);
751   set_boserif(0,2,6);
752
753   push_stroke((x3,y3)-(x3,y2),
754     (1.5,1.5)-(1.5,1.5));

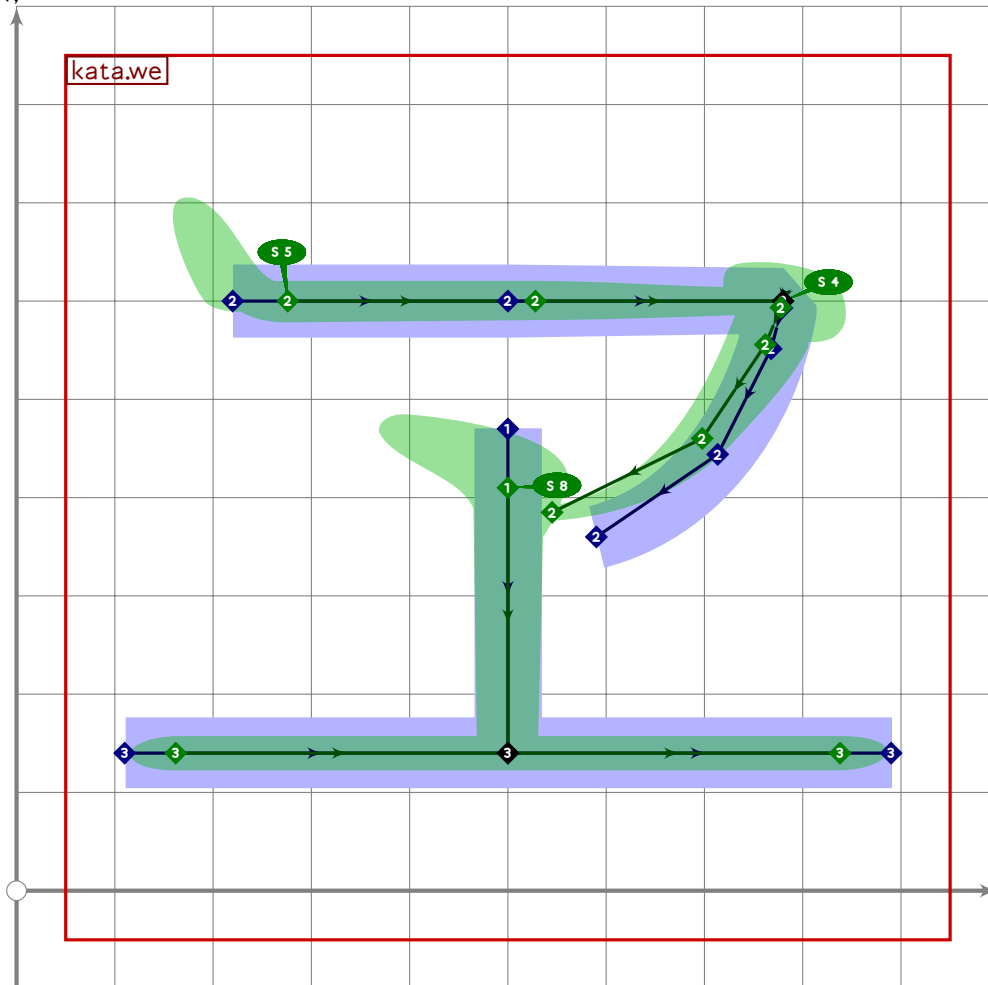
```

KATA

```

755
756 push_stroke((x4,y4)-(x4,0.5[y4,y1])-(x4,y1),
757   (0.75,2.65)-(1.4,1.4)-(1.6,1.6));
758 set_boserif(0,0,8);
759
760 push_stroke((x1,y2)-(x6,y2),
761   (0.7,3.3)-(1.8,1.8)-(0.7,3.3));
762 replace_strokep(0)(insert_nodes(oldp)(0.5));
763 set_boserif(0,0,5);
764 set_boserif(0,2,6);
765 expand_pbox;
766 enddef;

```



```

767
768 vardef kata.we =
769   push_pbox_toexpand("kata.we");
770
771   push_stroke((500,470-60*mincho)-(500,140),
772     (1.5,1.5)-(1.4,1.4));
773   set_boserif(0,0,8);
774
775   kata.fu_stroke((220,600),(780,600),

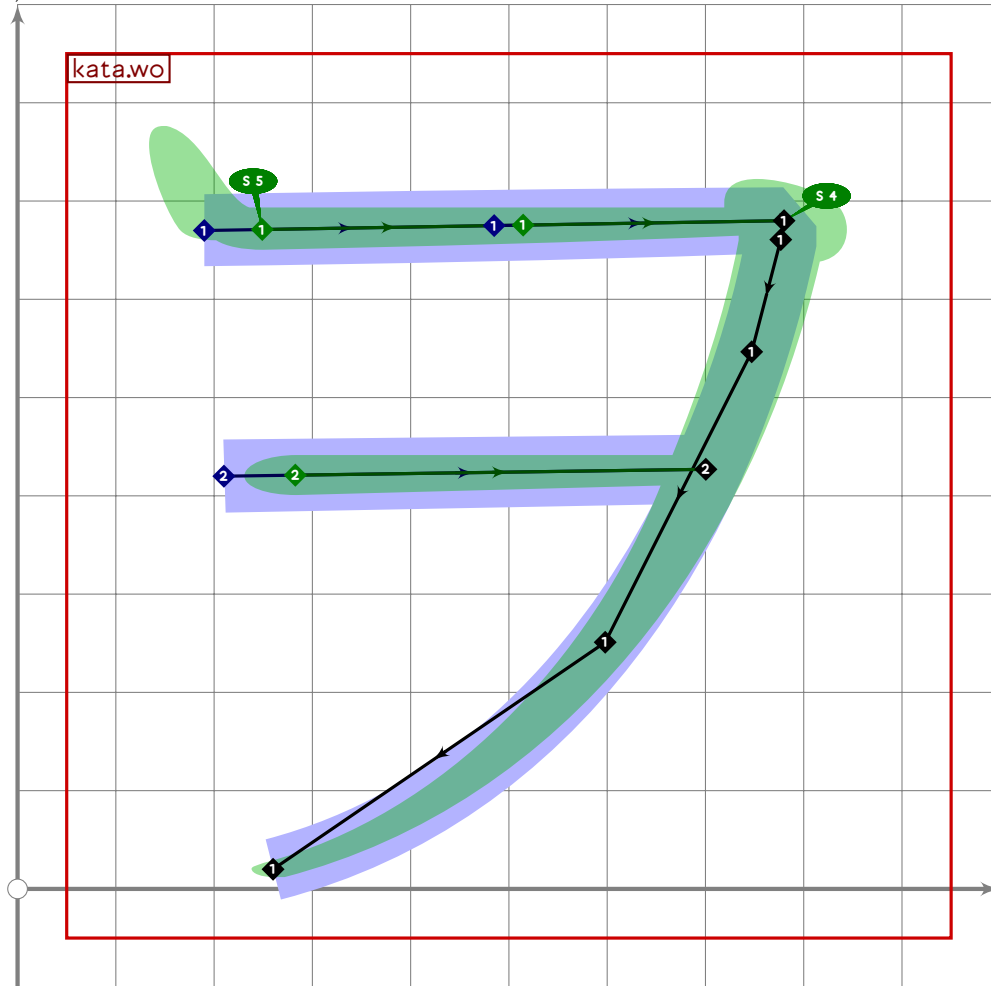
```

U+30F2
tsuku.uni30F2

```

776 (0.5*mincho)[(590,360),point 0 of get_strokep(0)];
777
778 push_stroke((110,140)–(500,140)–(890,140),
779 (0.7,2.9)–(1.7,1.7)–(0.7,2.9));
780 set_boserif(0,0,5);
781 set_boserif(0,2,6);
782 expand_pbox;
783 enddef;

```

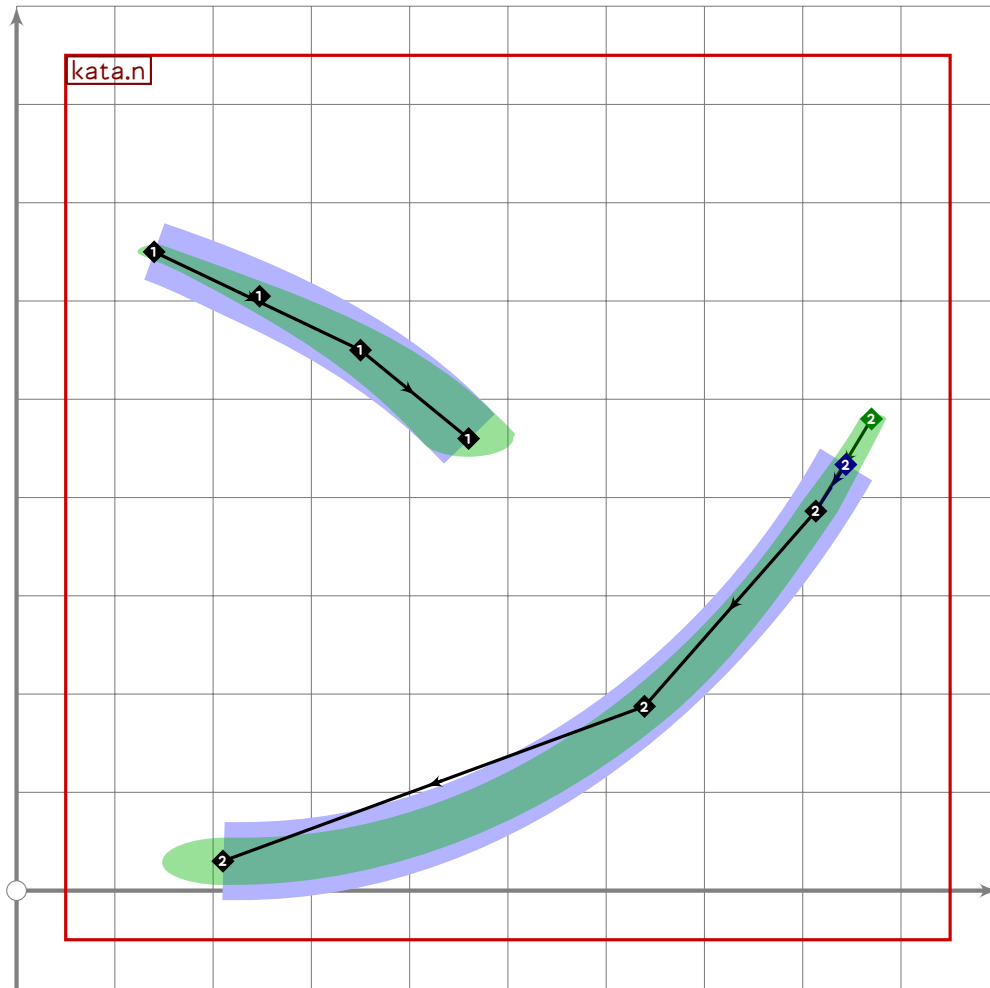


```

784
785 vardef kata.wo =
786   push_pbox_toexpand("kata.wo");
787
788   kata.fu_stroke((190,670),(780,680),(260,20));
789
790   z1=get_strokep(0) intersectionpoint ((0,420)–(1000,430));
791   push_stroke((210,420)–z1,
792 (0.7,3.3)–(1.6,1.6));
793   set_boserif(0,0,5);
794   expand_pbox;
795 enddef;

```

KATA

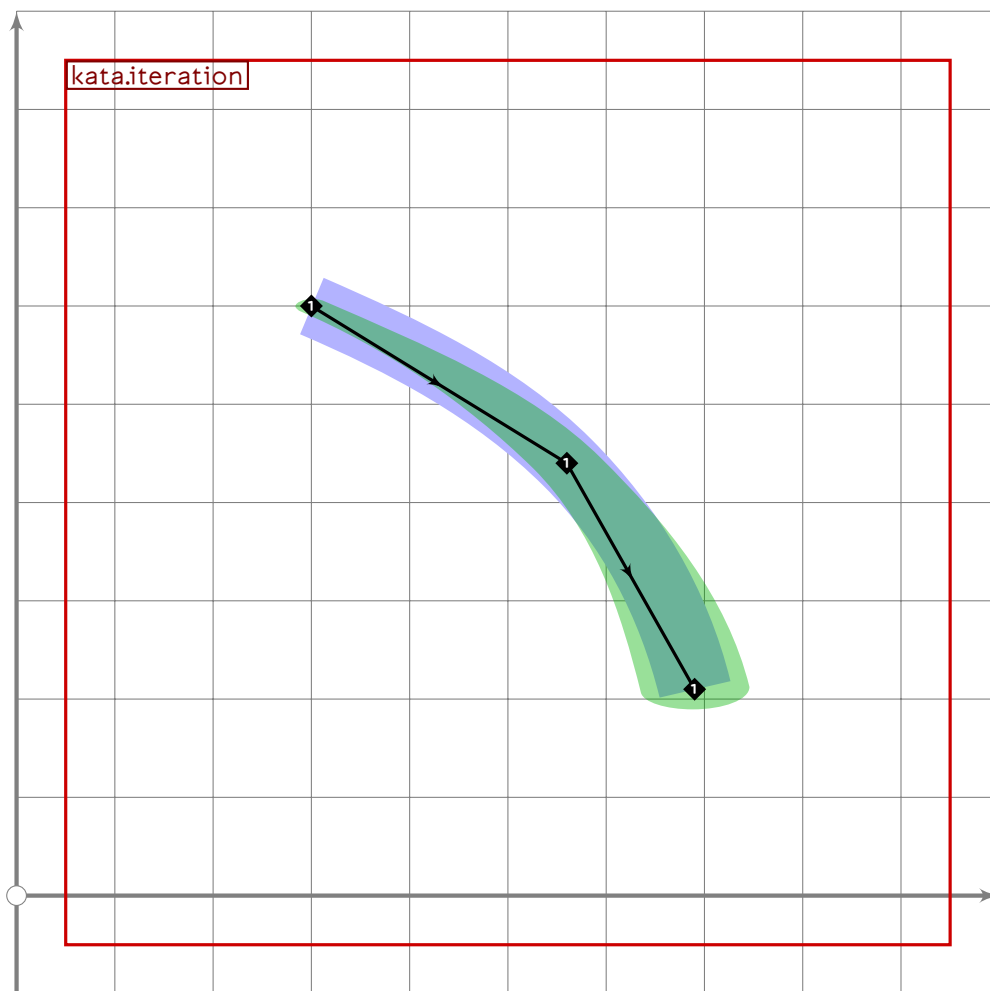


```

796
797 vardef kata.n =
798   push_pbox_toexpand("kata.n");
799
800   push_stroke((140,650)..tension 1.2..(350,550)..(460,460),
801     (1,1)..(1.6,1.6)..(1.8,1.8));
802
803   kata.no_stroke((870,480),(210,30));
804   replace_strokeq(0)((0.9,0.9)-(1.1,1.1)-(1.4,1.4)-(2.2,2.2));
805   expand_pbox;
806 enddef;

```

U+30FD
tsuku.uni30FD



```

807
808 vardef kata.iteration =
809   push_pbox_toexpand("kata.iteration");
810
811   push_stroke((300,600){curl 0.2}..(560,440)..(690,210),
812     (1,1)-(1.5,1.5)-(2,2));
813   expand_pbox;
814 enddef;

```

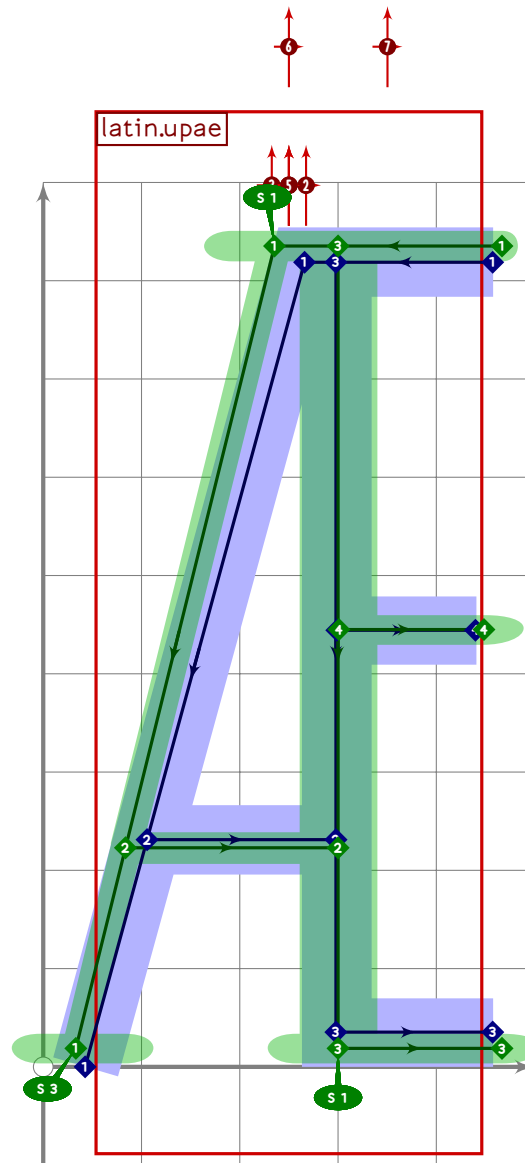
The diagram shows a stylized uppercase letter 'A' on a grid. The letter is composed of three strokes, each represented by a colored line with arrows indicating direction and numbered green circles indicating the start of a stroke. Stroke 1 (blue) starts at the bottom left, goes up to the top left, then down to the top right, and finally down to the bottom right. Stroke 2 (green) starts at the top left, goes right to the top right, then down to the bottom right, and finally up to the bottom left. Stroke 3 (red) starts at the top left, goes right to the top right, then down to the bottom right, and finally up to the bottom left. The letter is filled with a light blue color. There are also green oval shapes at the bottom left and bottom right, and a red oval shape at the top right. The text 'latin.upa' is in the top left corner.

LATI


```

43  z5=whatever[(z2+alternate_adjust*right/2),z3]-(2,0);
44  y4=y5=vmetric(0.333);
45
46  push_stroke(z1-(z2+alternate_adjust*left/2),(1.6,1.6)-(1.6,1.6));
47  set_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(ypart 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;

```



```

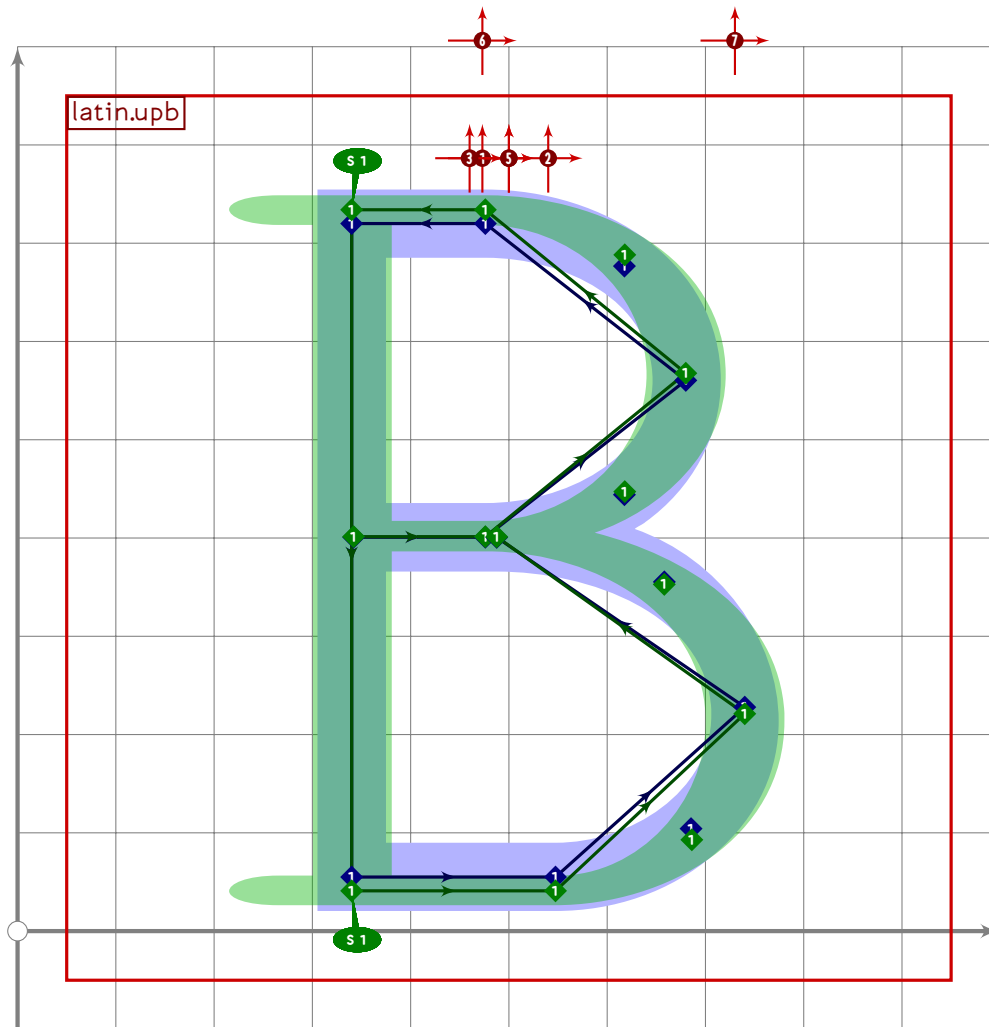
77
78 vardef latin.upae =
79   push_pbox_toexpand("latin.upae");
80   y1=y2=latin_wide_high_h;
81   y3=y4=latin_wide_low_h;
82   y5=y6=vmetric(0.522);
83
84   (x1+x7)/2=500;
85   x2=x3;
86   x1=x4;
87   x5=x2+2;
88   x6=0.89[x2,x1];
89   (x1-x2)=(y2-y3)*0.55;
90
91   y7=latin_wide_low_v;
92   x7=(-1.6)[x2,x1];

```

```

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));
135
136   set_botip(0,4,1);
137   set_botip(0,5,1);

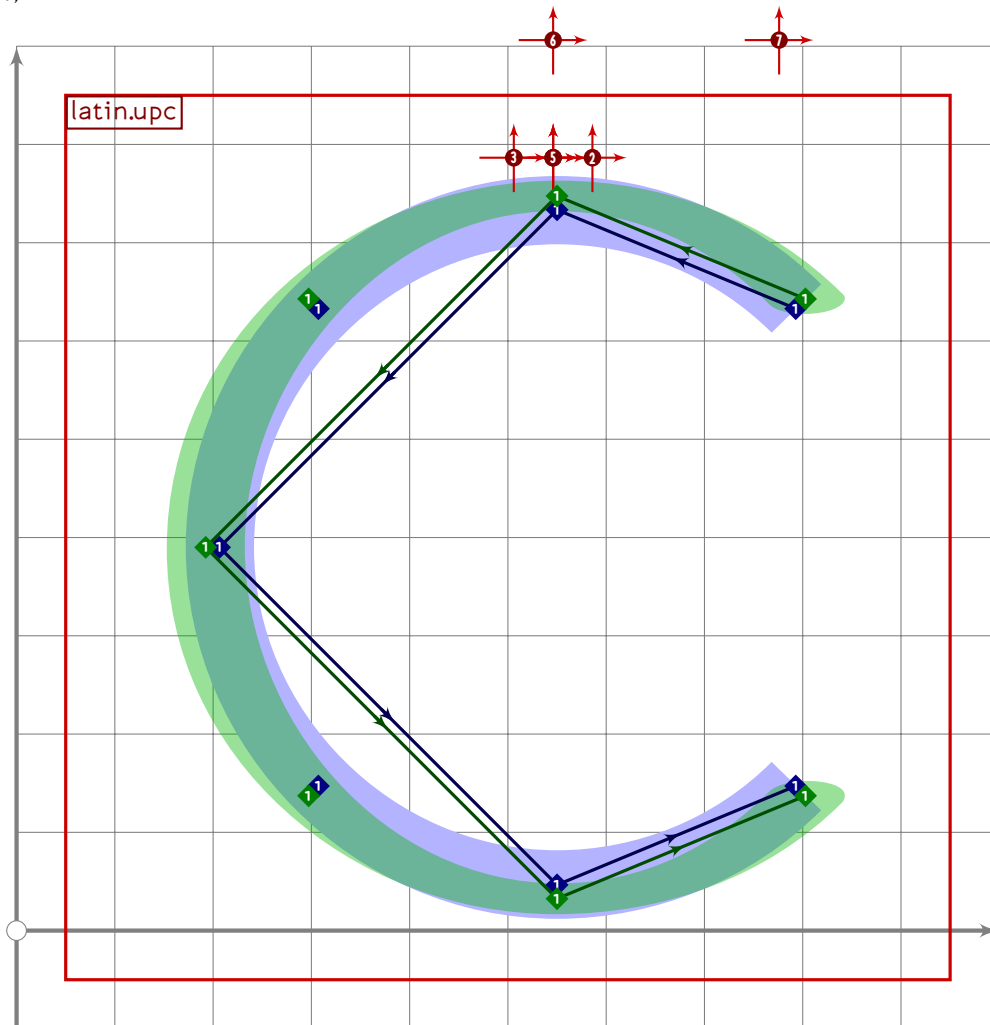
```

U+FF23
tsuku.uniFF23

```

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 enddef;

```



```

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)-

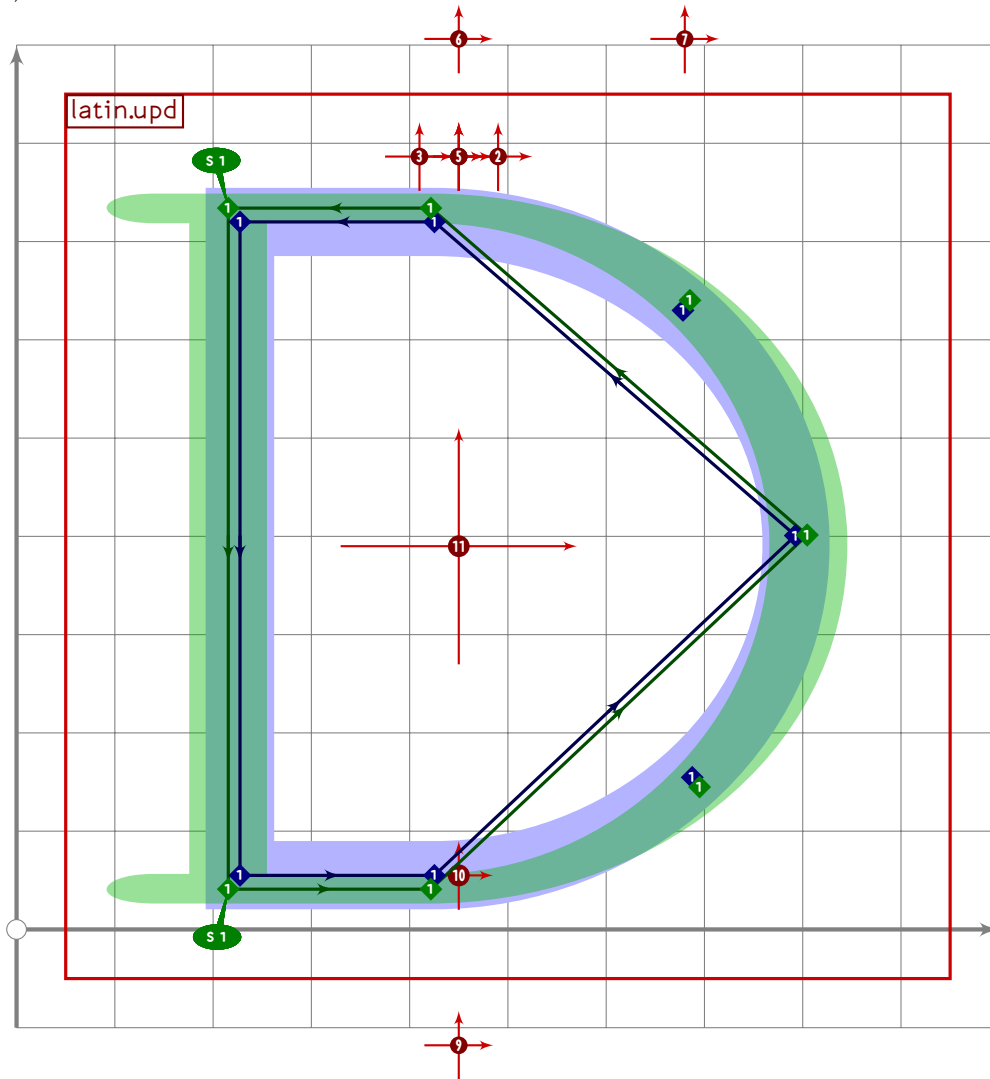
```

LATI

```

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

```



```

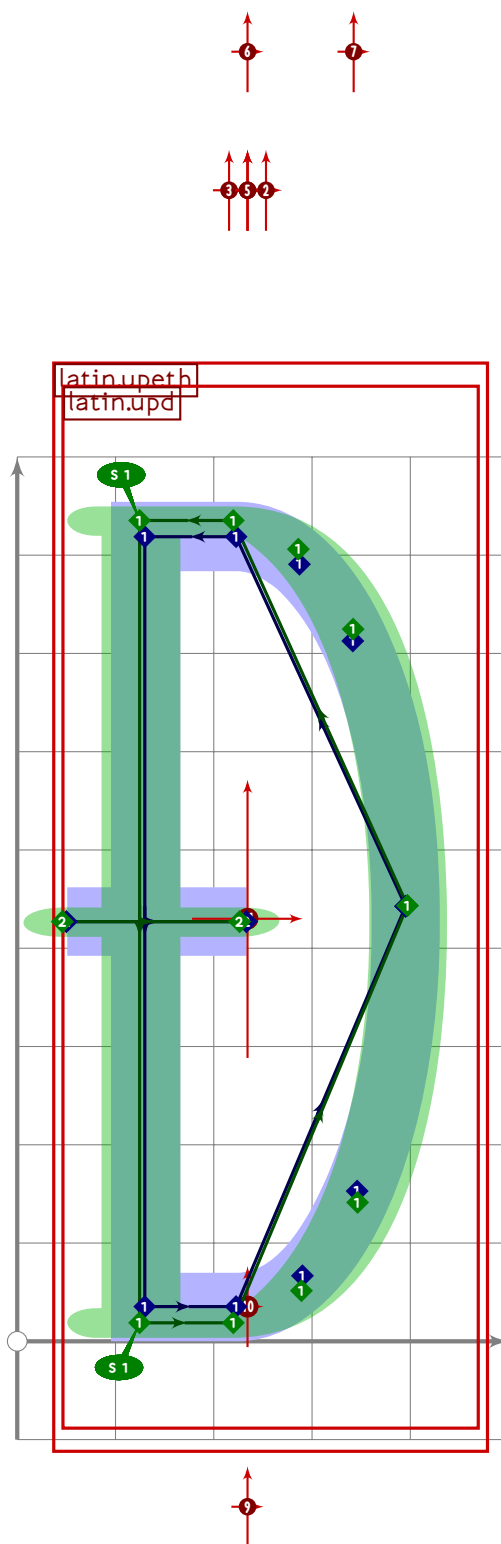
161
162 vardef latin.upd =
163   push_pbox_toexpand("latin.upd");
164   y1=y5=latin_wide_high_h;
165   y2=y3=latin_wide_low_h;
166   y4=0.52[y2,y1];
167
168   (x1+x4)/2=510;
169   (x4-x1)=0.85*(y1-y2);
170   x1=x2;
171   x3=x5=0.35[x1,x4];
172
173   push_stroke(z4.{left}z5-z1-z2-z3{right}..cycle,
174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

```

```

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;

```



```

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));

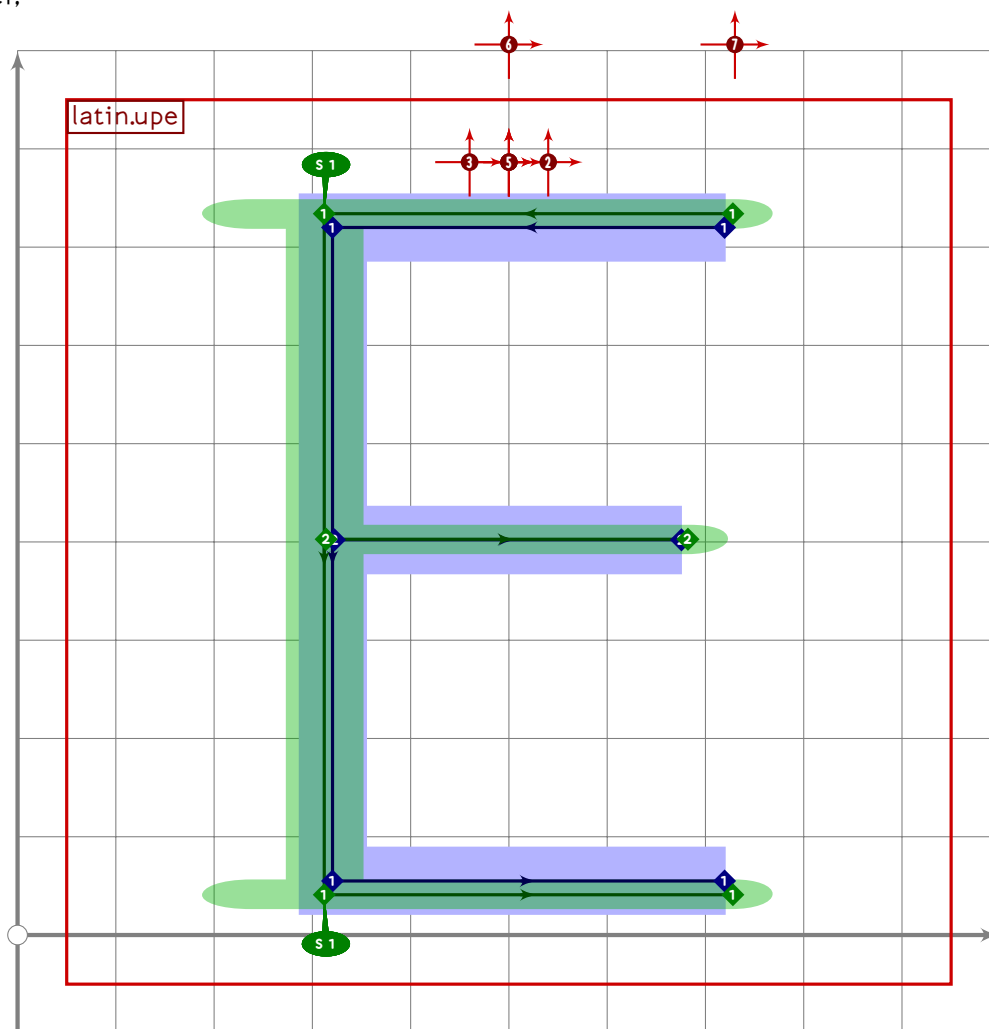
```


U+FF25
tsuku.uniFF25

```

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 endif;

```



```

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

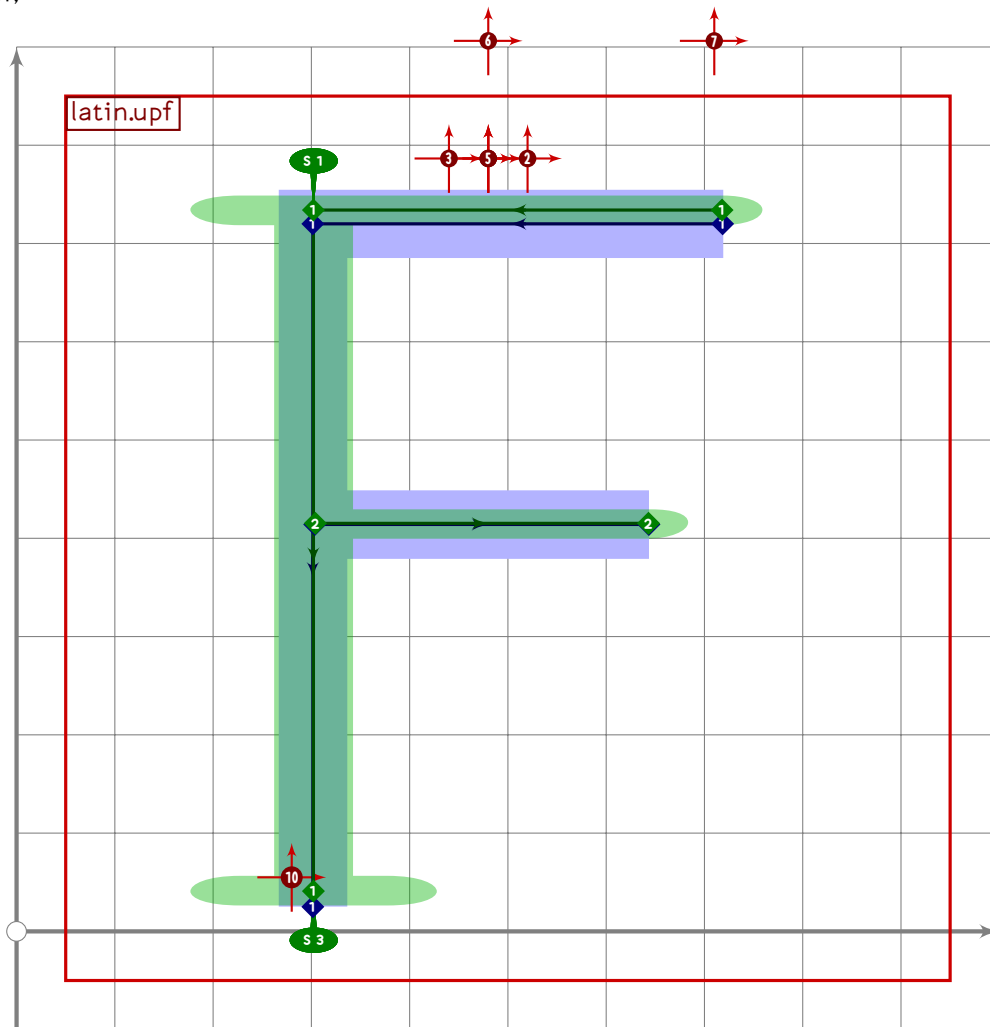
```

LATI

```

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;

```



```

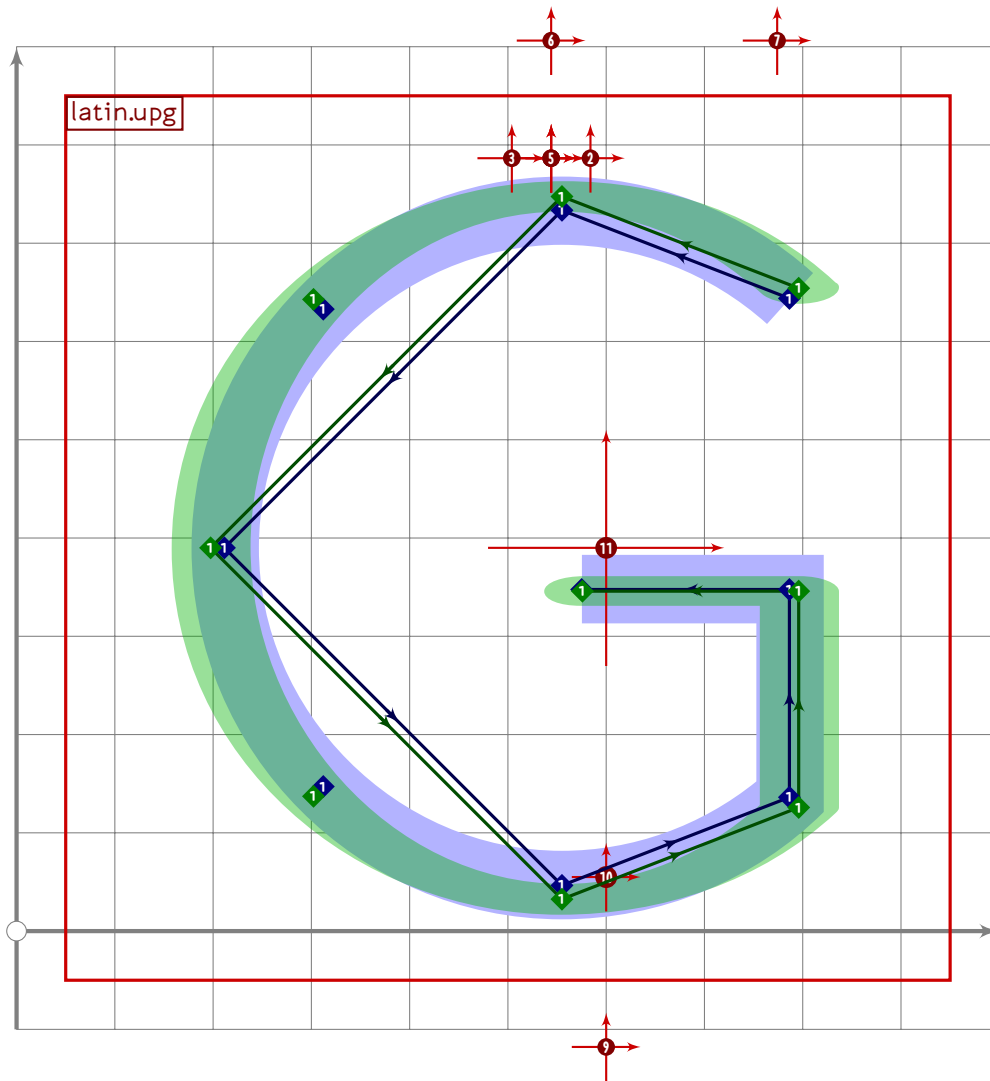
225
226 vardef latin.upf =
227   push_pbox_toexpand("latin.upf");
228   y1=y2=latin_wide_high_h;
229   y3=latin_wide_low_v;
230   y4=y5=vmetric(0.54);
231

```

```

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))

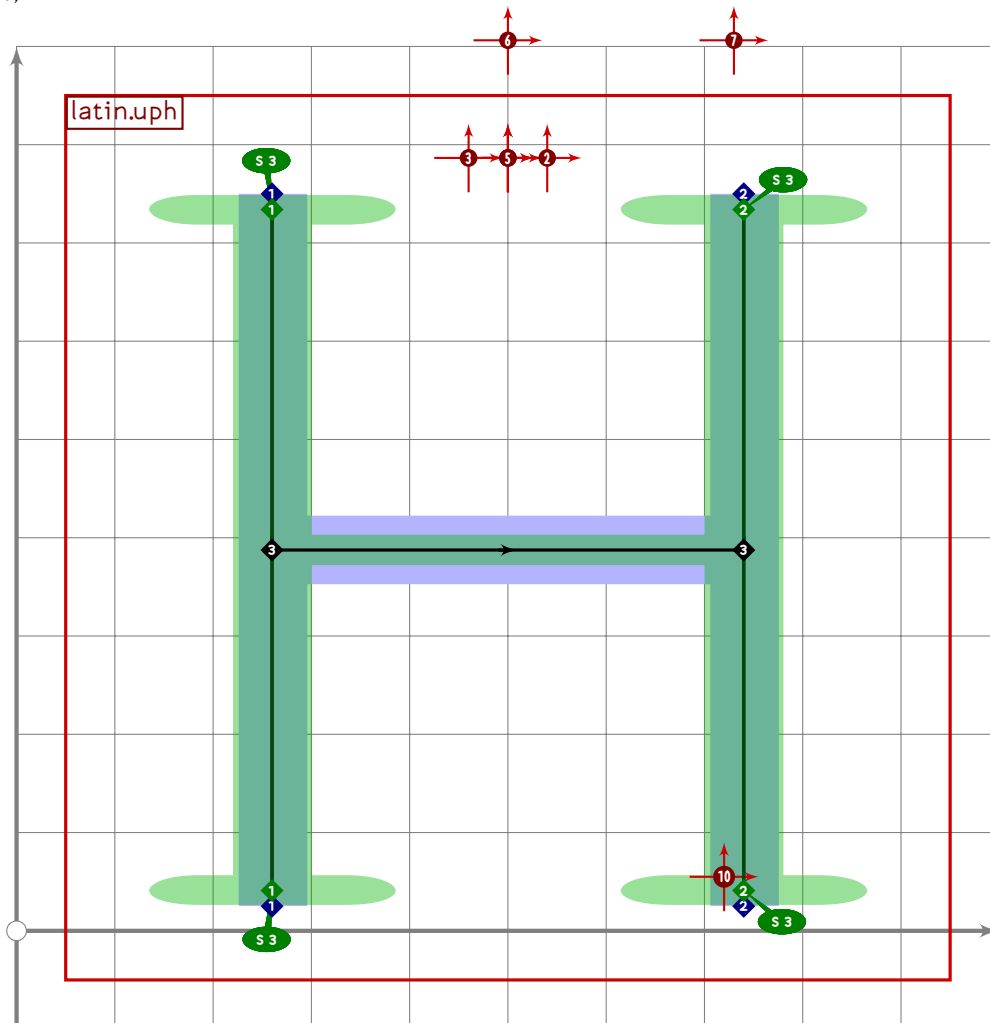
```

U+FF28
tsuku.uniFF28

```

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);

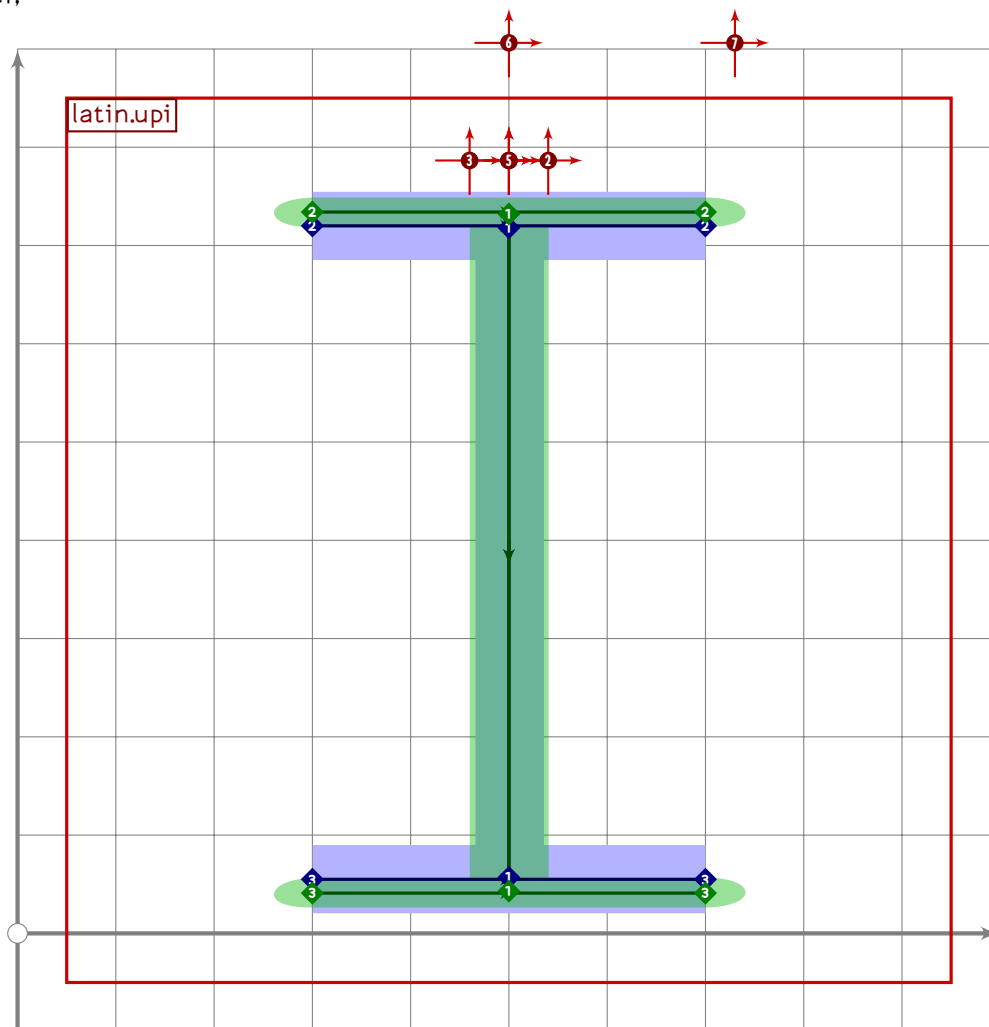
```

LATI

```

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(y part 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;

```



```

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),

```

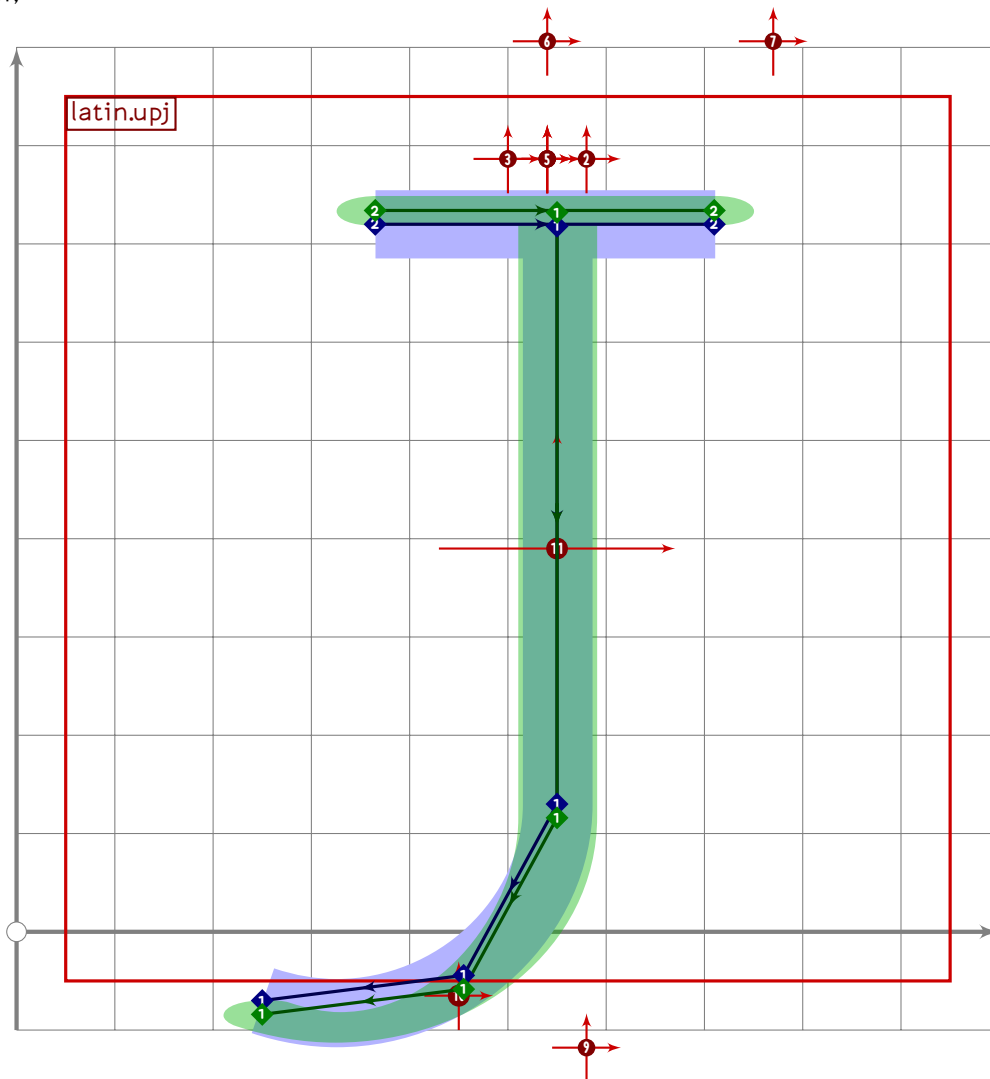
LATI

U+FF2A
tsuku.uniFF2A

```

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;

```



LATI

```

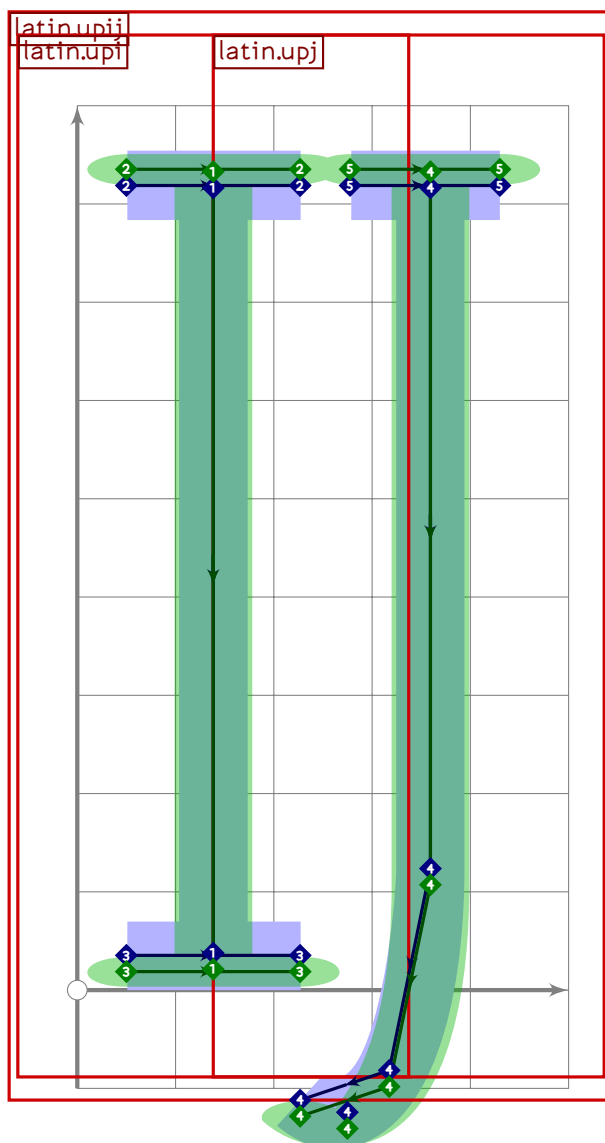
321
322 vardef latin.upj =
323   push_pbox_toexpand("latin.upj");
324   z1=(550,latin_wide_high_h-2);
325   z2=(550,latin_wide_low_h+75);
326   z3=z2+(-300,200);
327
328   push_stroke((z1–z2)..{curl 0.8}z3,

```

```

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;

```



```

344
345 vardef latin.upij =

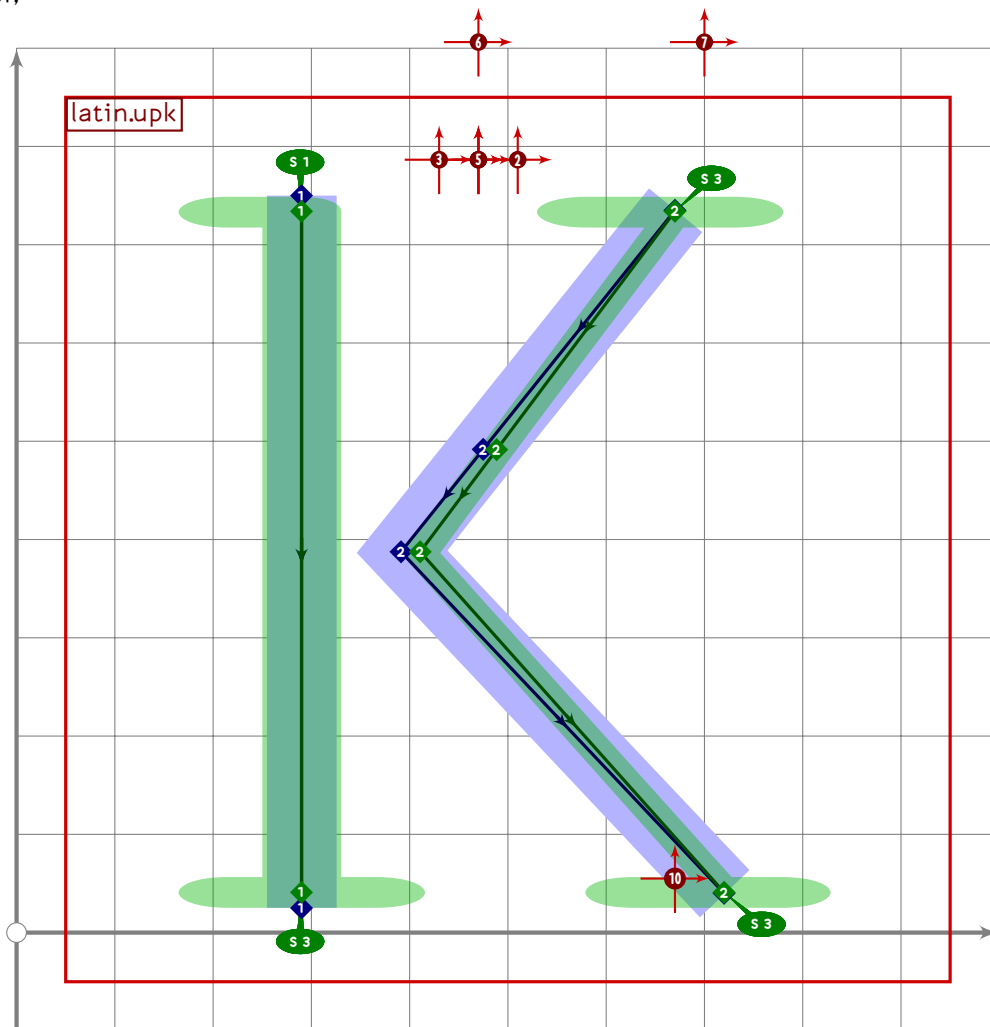
```


U+FF2B
tsuku.uniFF2B

```

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 enddef;

```



```

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));

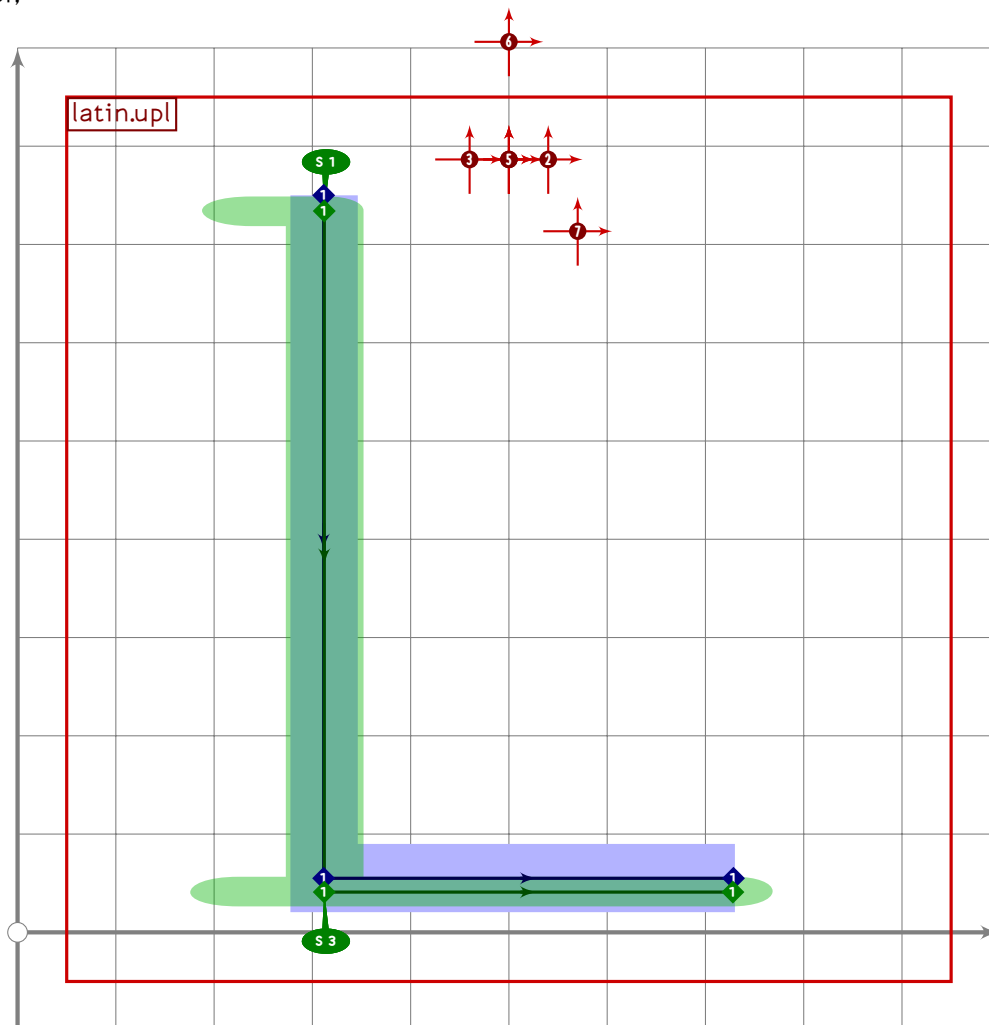
```

LATI

```

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;

```



```

384
385 vardef latin.upl =

```

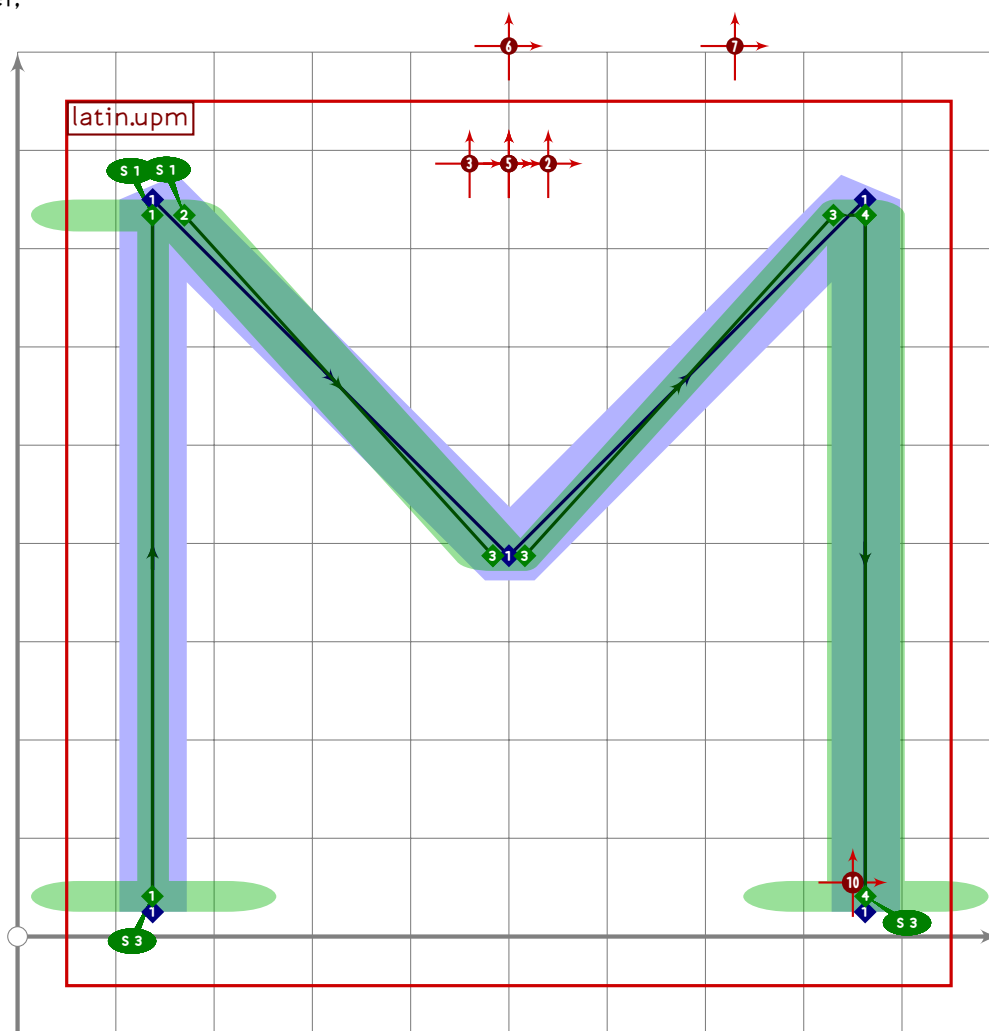
LATI

U+FF2D
tsuku.uniFF2D

```

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 endif;

```



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

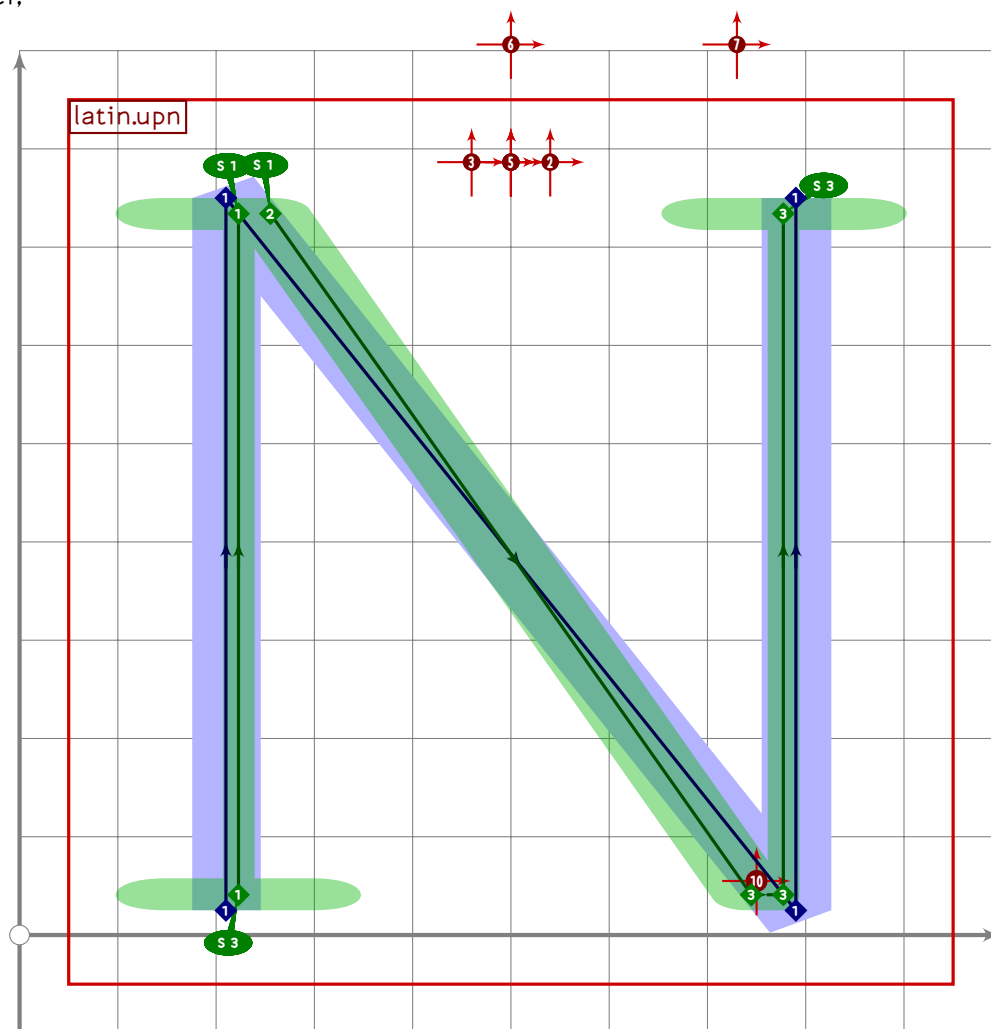
```

U+FF2E
tsuku.uniFF2E

```

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);

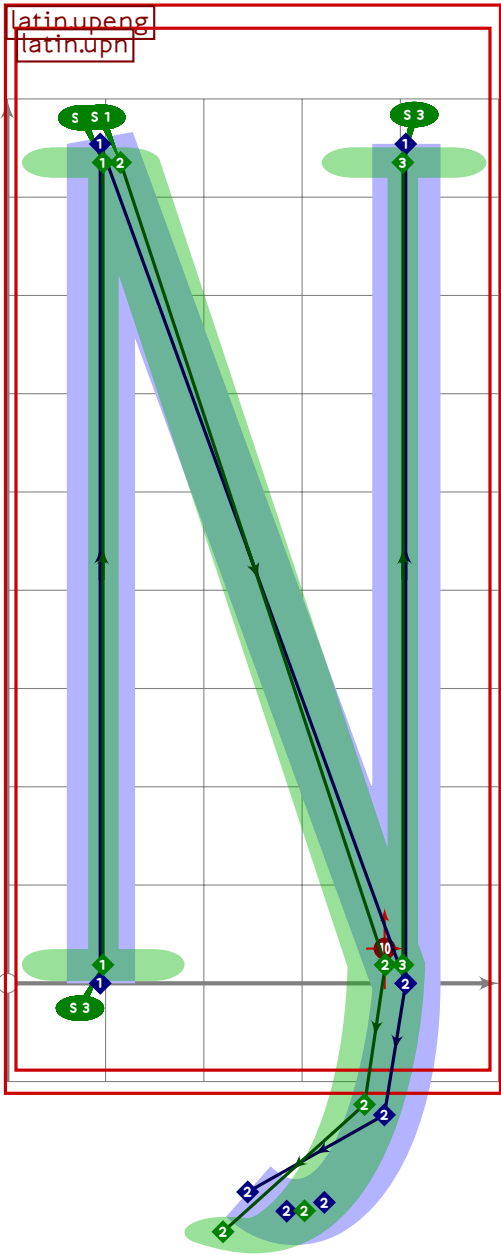
```

LATI

```

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;

```



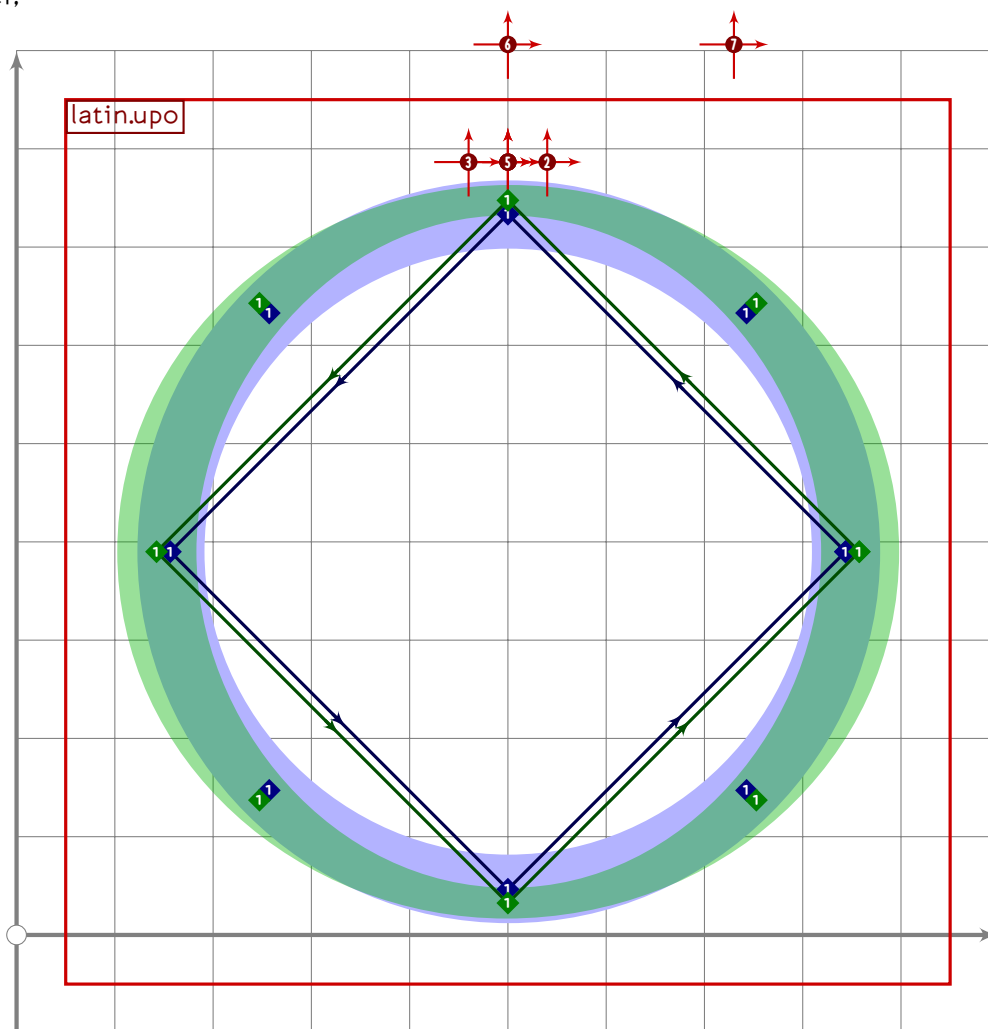
LATI

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

```

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(

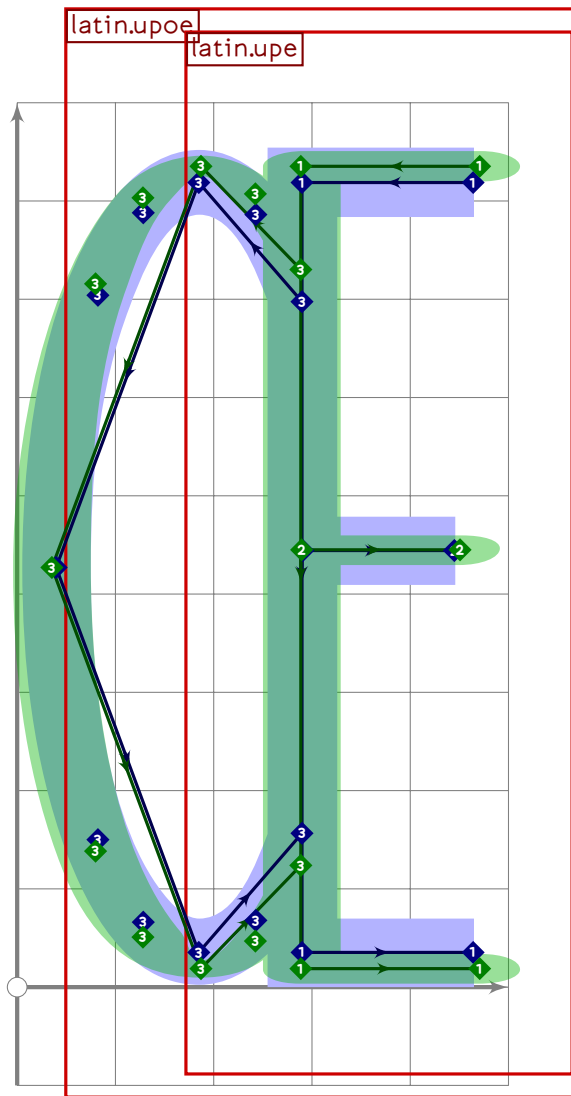
```


U+0152
tsuku.OE

```

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;

```



LATI

```

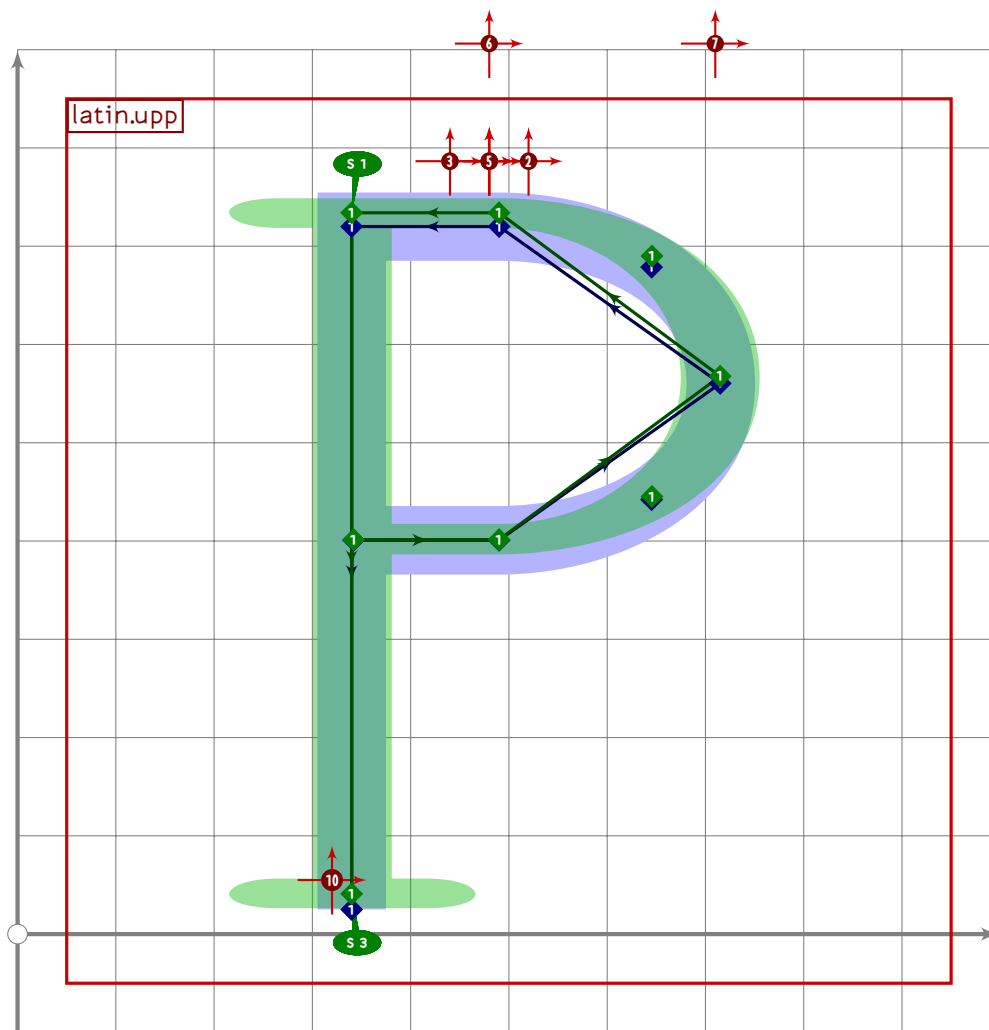
535
536 vardef latin.upoe =
537   push_pbox_toexpand("latin.upoe");
538   tsu_xform(identity shifted (280,0))(latin.upe);
539   set_boserif(-1,1,whatever);
540   set_boserif(-1,2,whatever);
541   push_stroke(
542     ((1,0)..(0,1)..(-1,0)..(0,-1)..(1,0))

```

```

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;

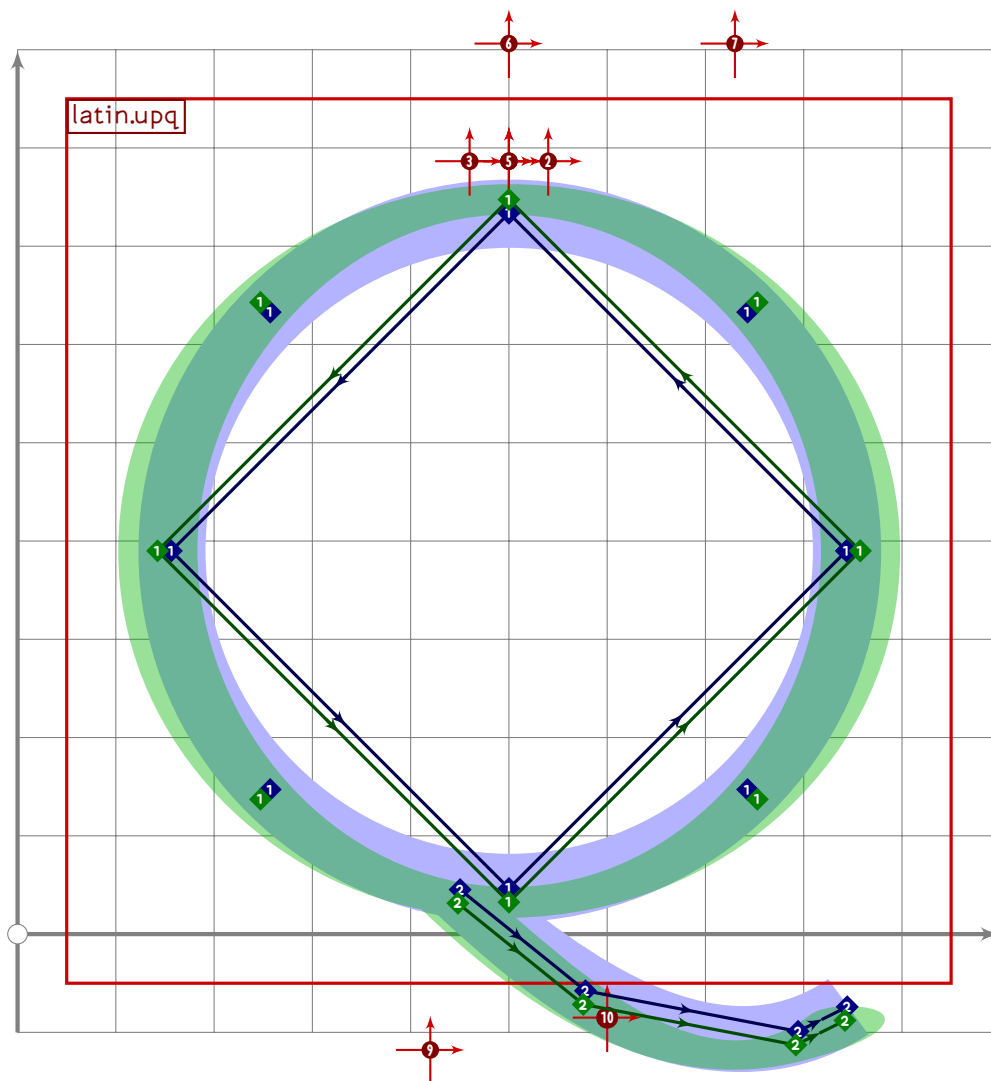
```



```

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;

```



```

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(oldp)(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));

```

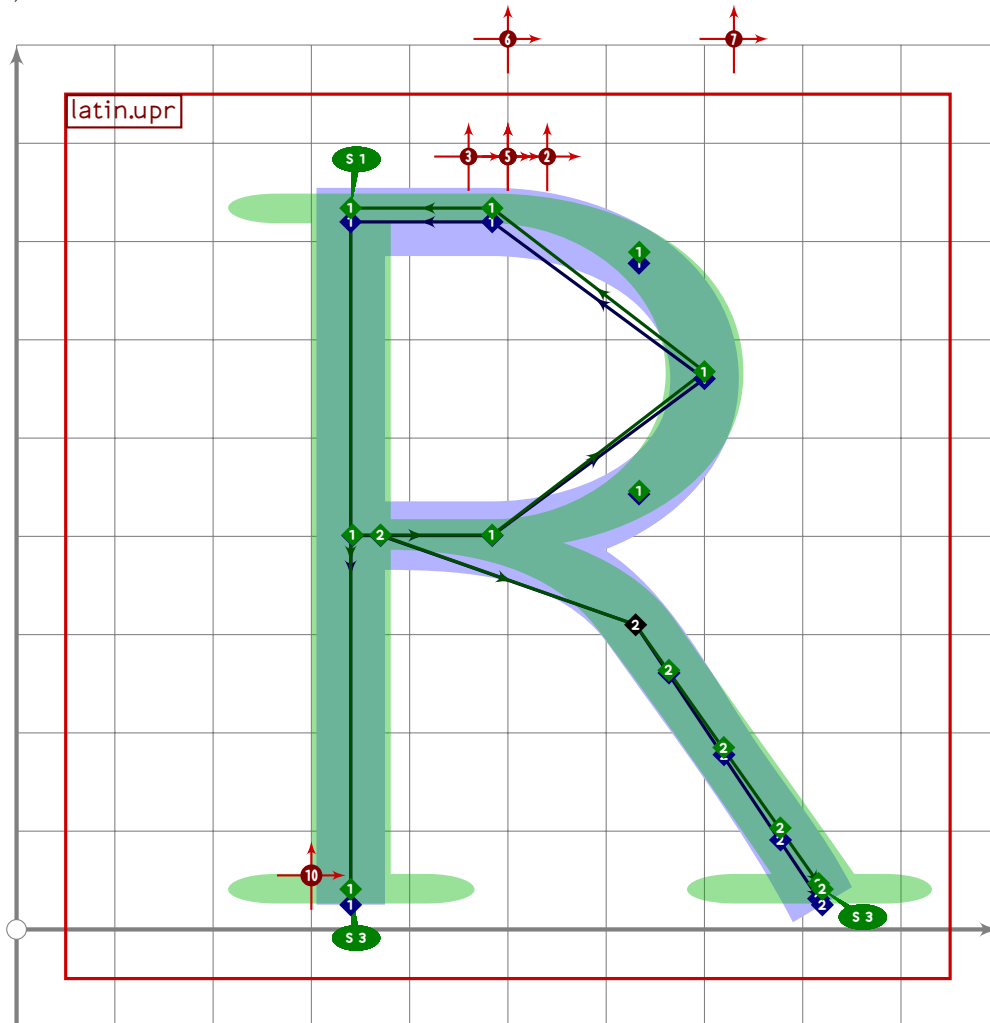
LATI

U+FF32
tsuku.uniFF32

```

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.upr_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_strokep(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))

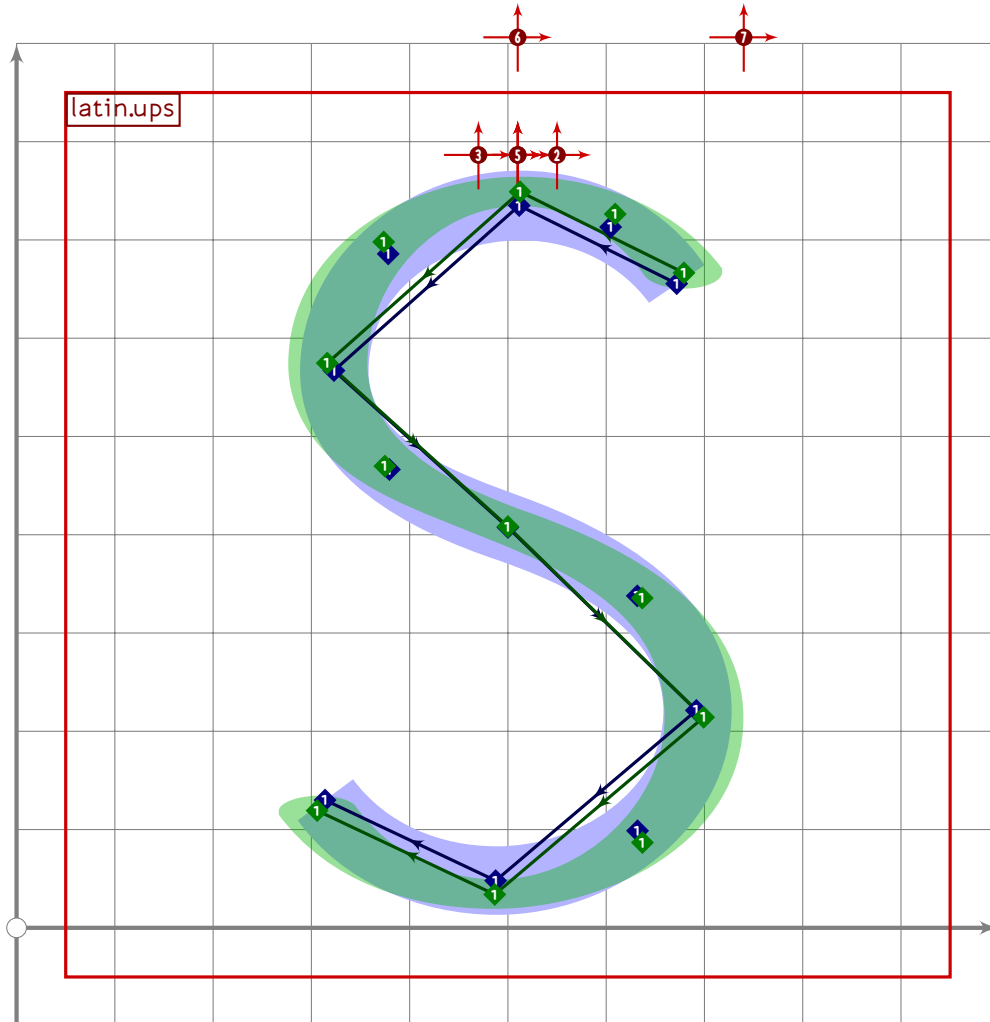
```

LATI

```

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 endif;

```



```

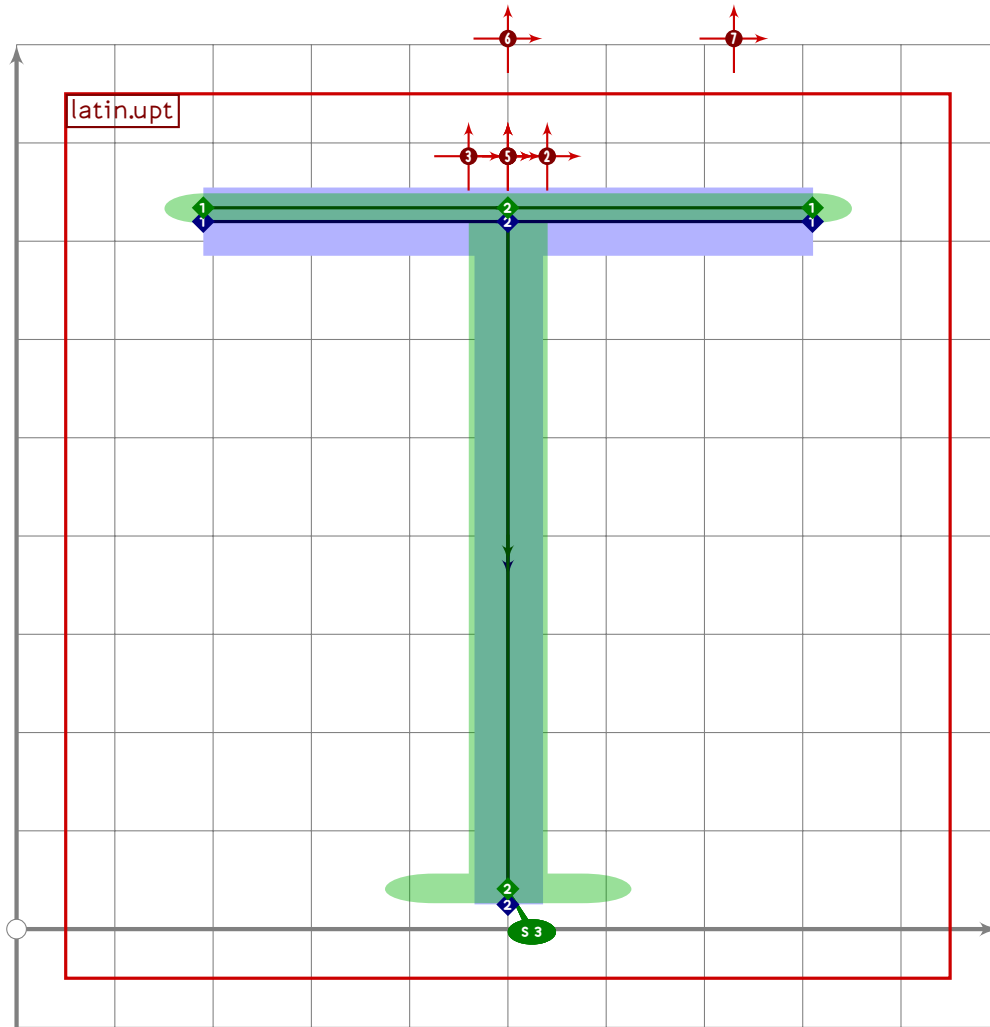
630
631 vardef latin.ups =
632   push_pbox_toexpand("latin.ups");
633   transform tatb;
634   path mycurve;
635
636   mycurve:=(1,0)..(0,1)..(-1,0);
637
638   y2=latin_wide_high_r;
639   y0=y3=vmetric(0.77);
640   y4=vmetric(0.53);
641   y5=y8=vmetric(0.25);
642   y6=latin_wide_low_r;
643
644   0.48[x1,x7]=0.48[x2,x6]=0.48[x3,x5]=x4=500;

```

```

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;

```

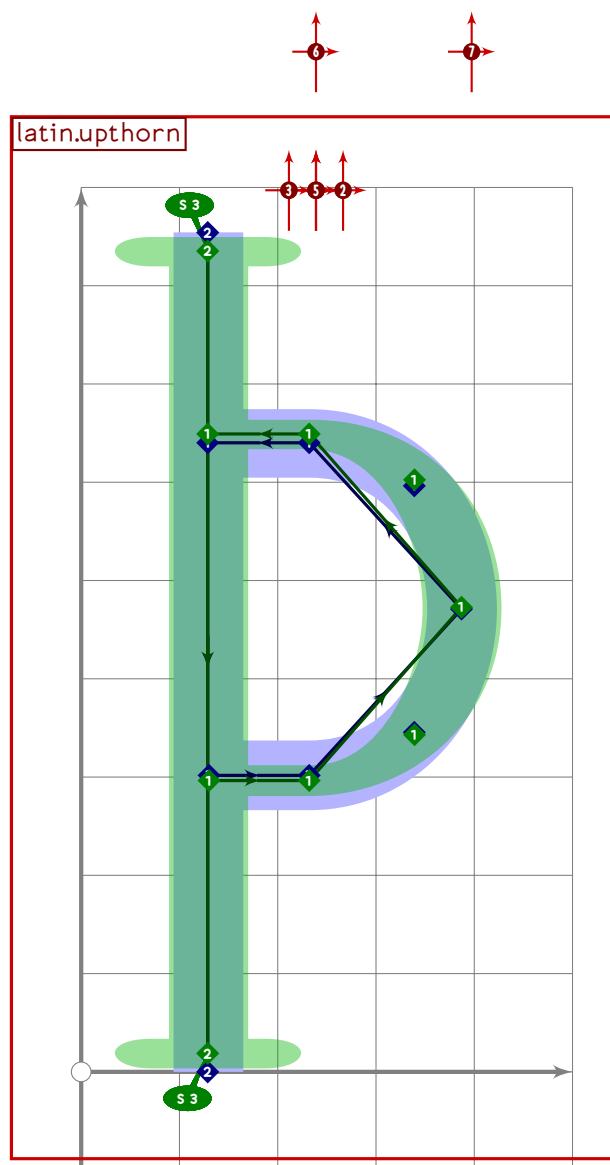


```

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;

```


U+00DE
tsuku.Thorn



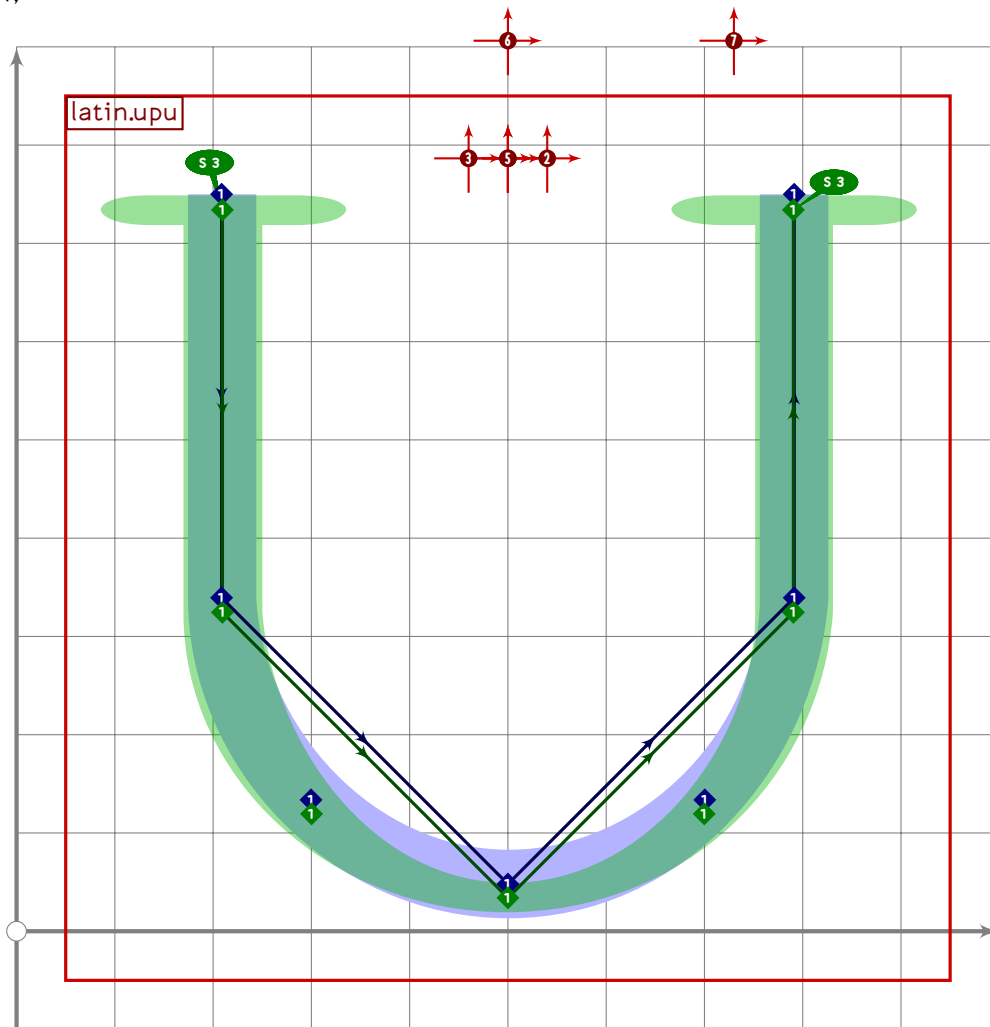
```

688
689 vardef latin.upthorn =
690   push_pbox_toexpand("latin.upthorn");
691   latin.upp_base(375);
692   replace_strokep(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_strokep(0),latin_wide_high_v)
696     -(xpart point infinity of get_strokep(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]));

```

LATI

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



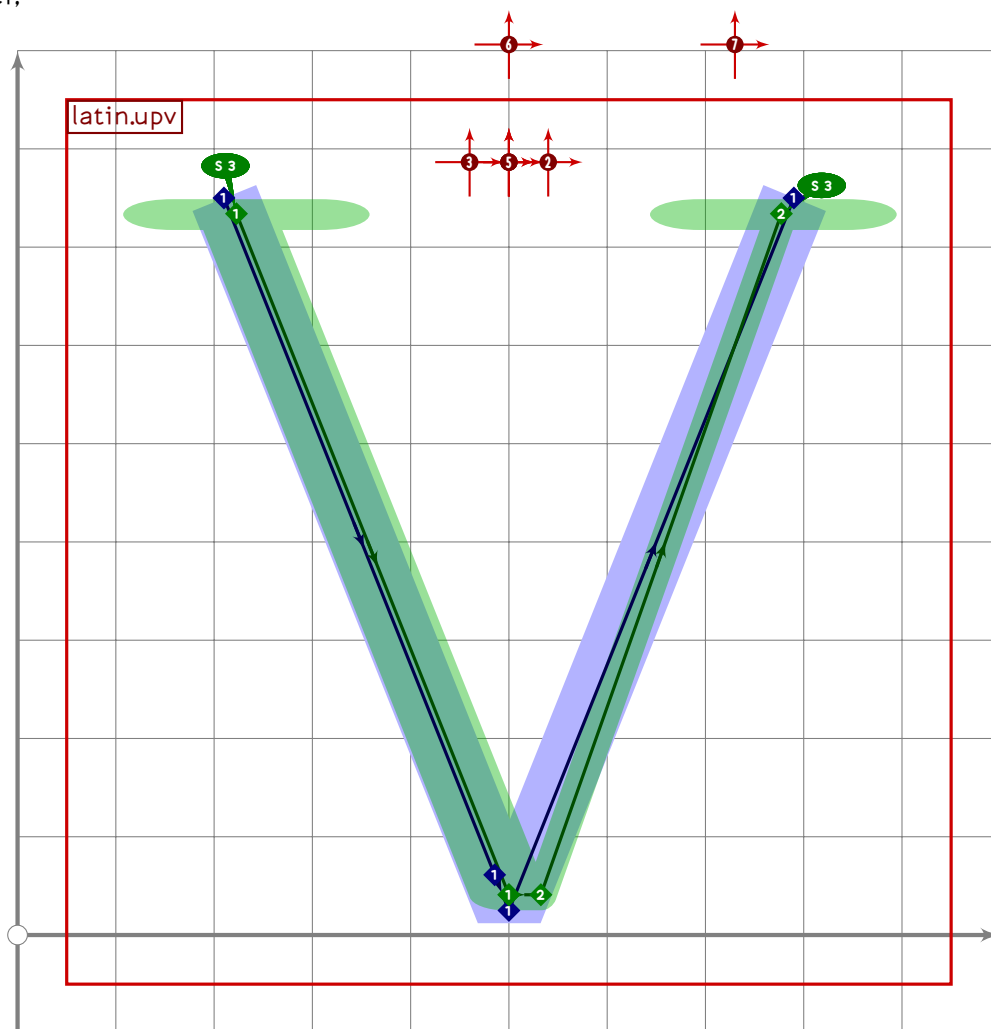
```
706
707 vardef latin.upu =
708   push_pbox_toexpand("latin.upu");
709   x1=x2;
710   x3=500;
711   x4=x5;
712   (x1+x5)/2=x3;
713   (x5-x1)=(y1-y3)*0.83;
714
715   y1=y5=latin_wide_high_v;
716   y2=y4;
717   y2-y3=x3-x2;
718   y3=latin_wide_low_r;
719
720   push_stroke(z1-z2{dir 274}..z3.{dir 86}z4-z5,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

U+FF36
tsuku.uniFF36

```

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

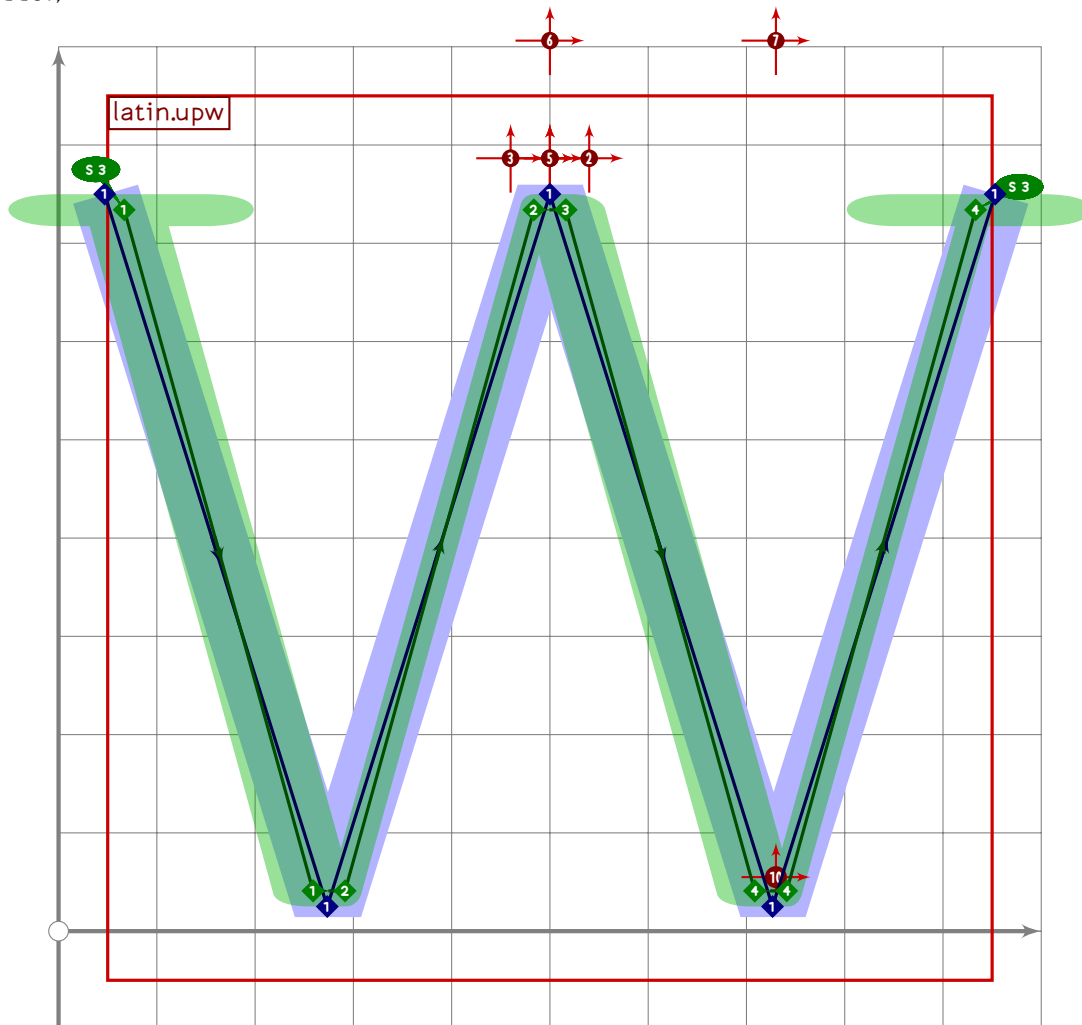
```

LATI

```

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");

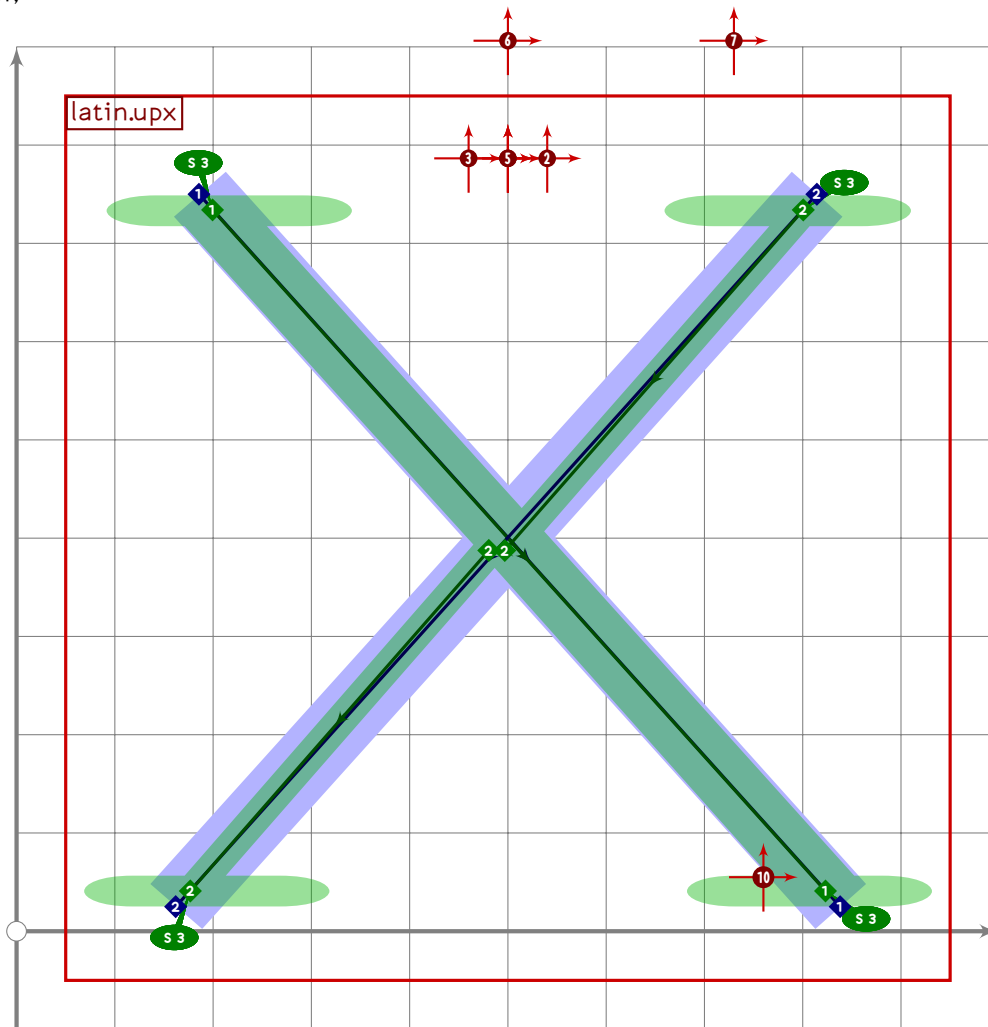
```

```

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));

```

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



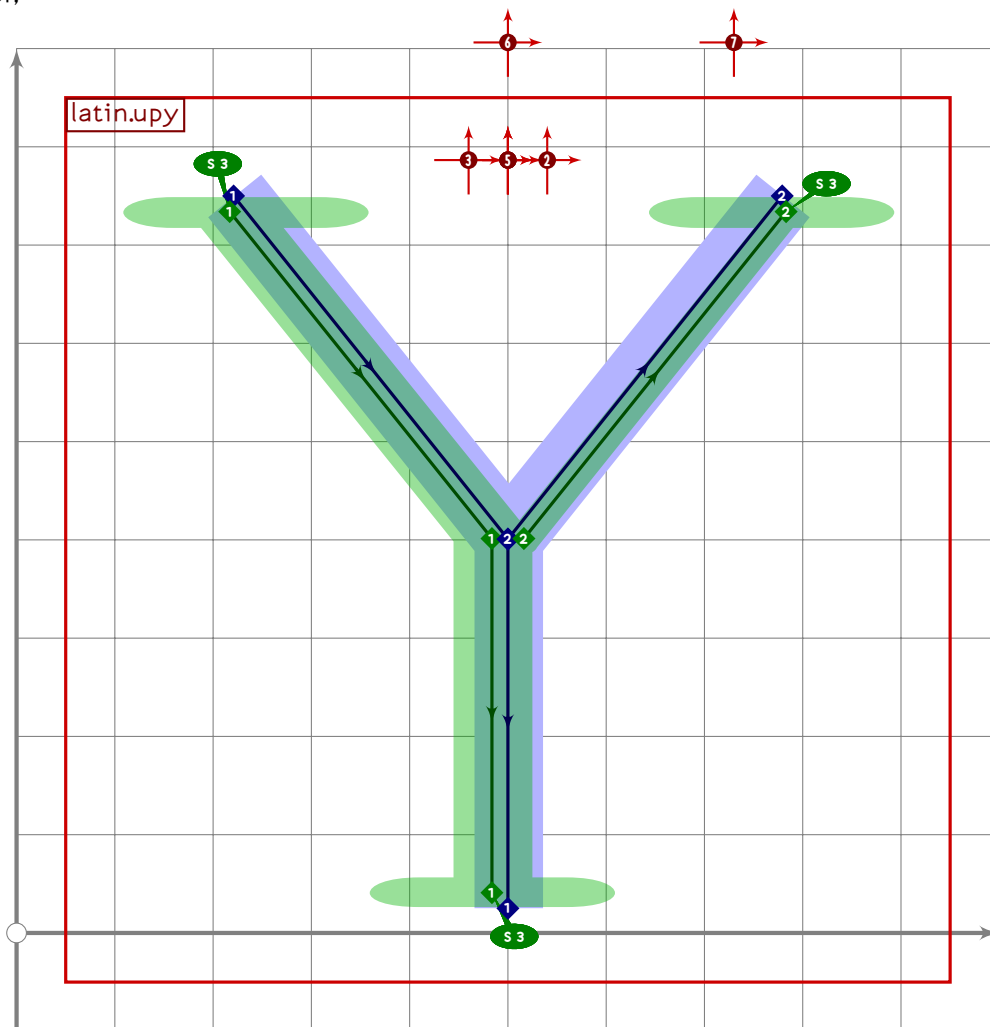
```
814
815 vardef latin.upx =
816   push_pbox_toexpand("latin.upx");
817   (x1+x3)/2=500;
818   (x2+x4)/2=500;
819   (x2+x3-x1-x4)=((y1-y2)*0.9)*2;
820   (x3-x1)=(x2-x4)*0.93;
821
822   y1=y3=latin_wide_high_v;
823   y2=y4=latin_wide_low_v;
824
825   push_stroke(z1-z2,(1.6,1.6)-(1.6,1.6));
826   set_boserif(0,0,3);
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,
```

U+FF39
tsuku.uniFF39

```

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;

```



LATI

```

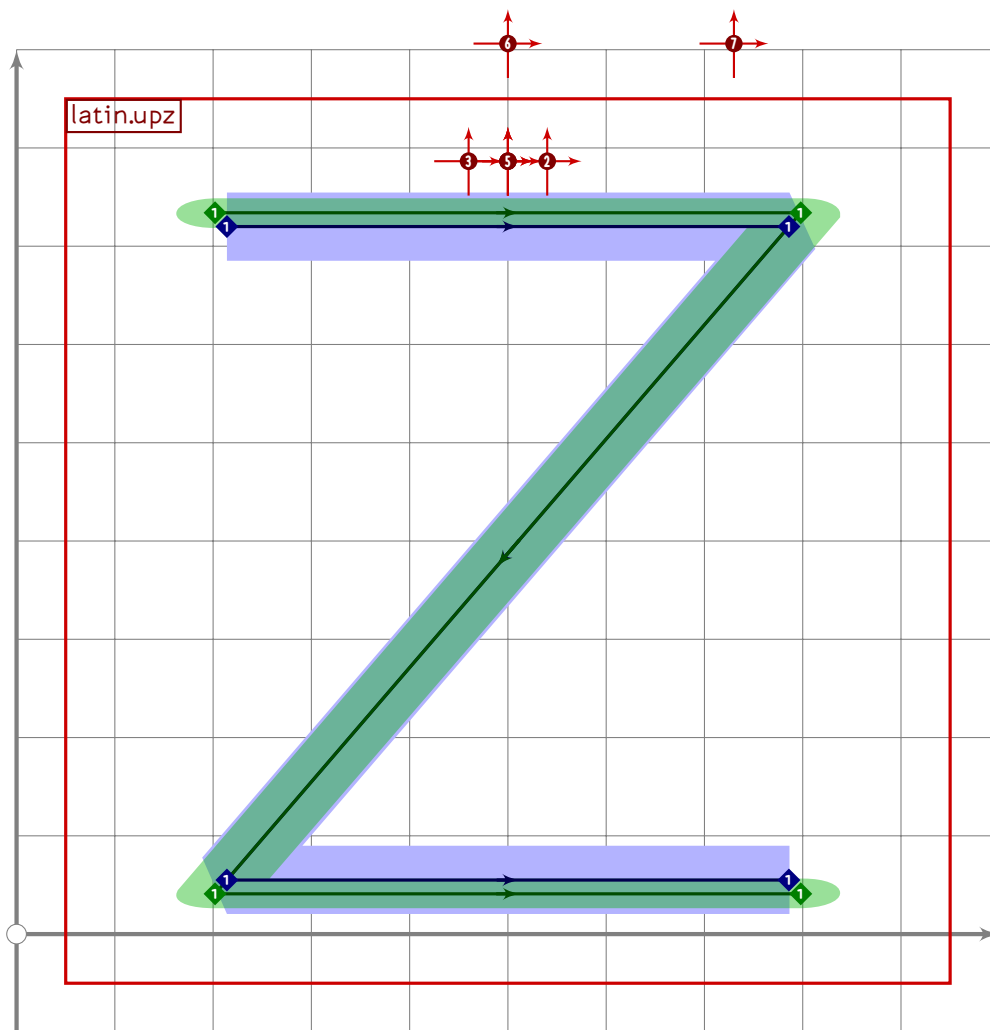
848
849 vardef latin.upy =
850   push_pbox_toexpand("latin.upy");
851   (x3+x1)/2=x2=x4=if do_alteration: 500-alternate_adjust/2 else: 500 fi;

```

```

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;

```

```

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;

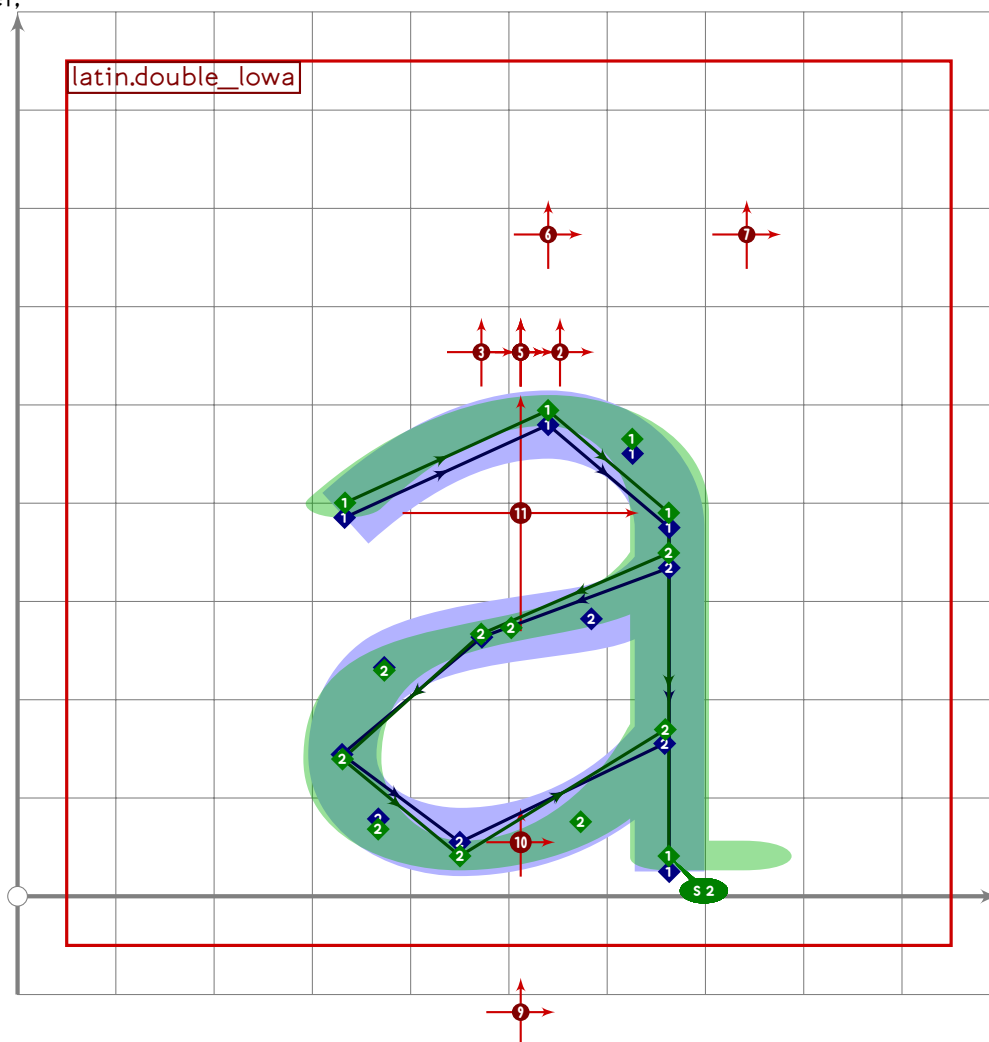
```

U+FF41
tsuku.uniFF41

```

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;

```

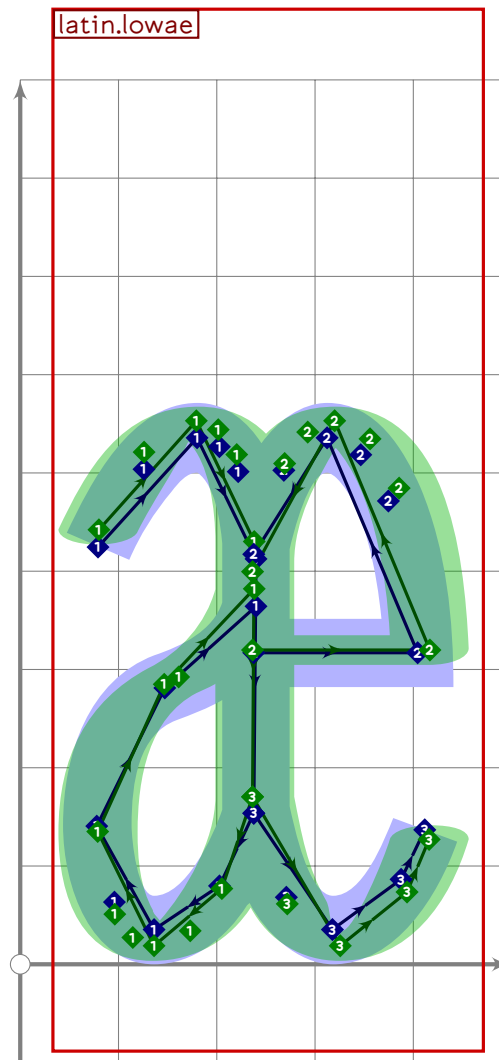


LATI

```

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

```



```

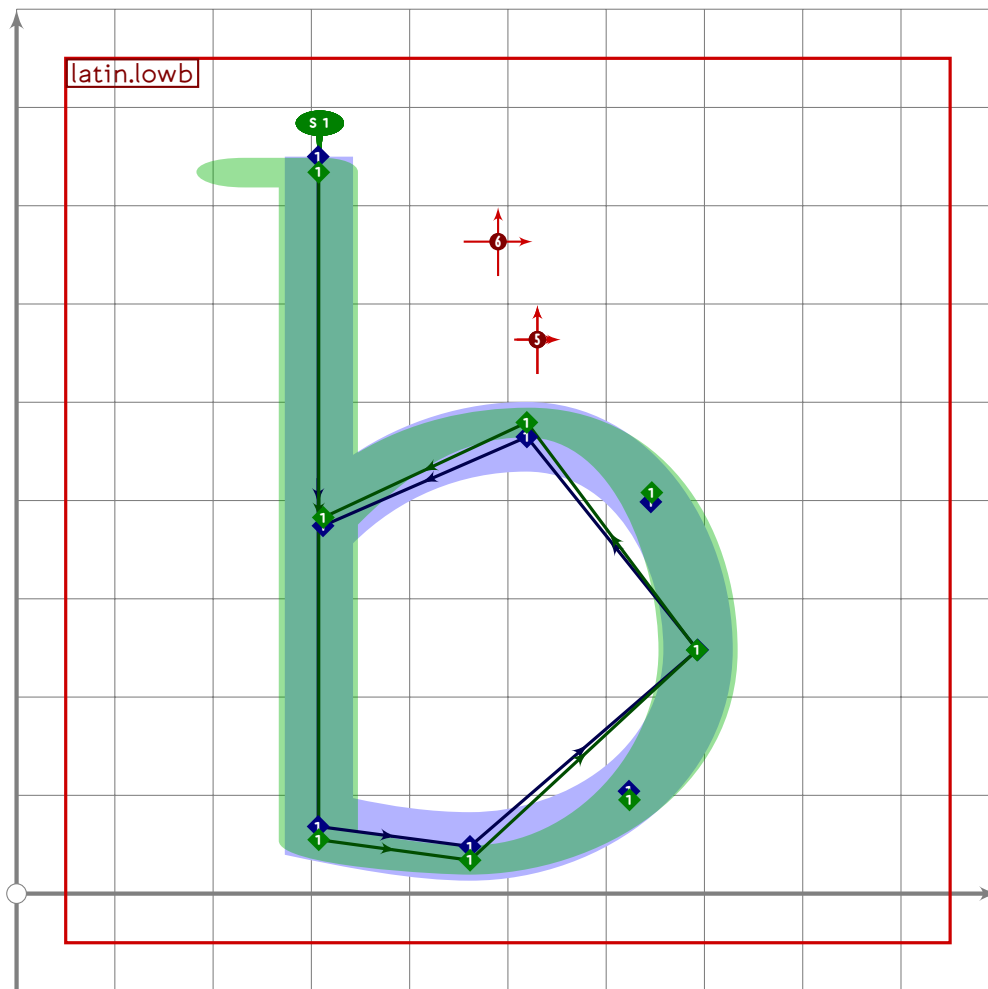
964
965 vardef latin.lowae =
966   push_pbox_toexpand("latin.lowae");
967   begingroup
968     save saved_sp;
969     save astem,abowl,astroke,estroke,etop,ebot;
970     path astem,abowl,astroke,estroke,etop,ebot;
971     saved_sp:=sp;
972
973     latin.double_lowae;
974     set_boserif(-1,3,whatever);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

```



```

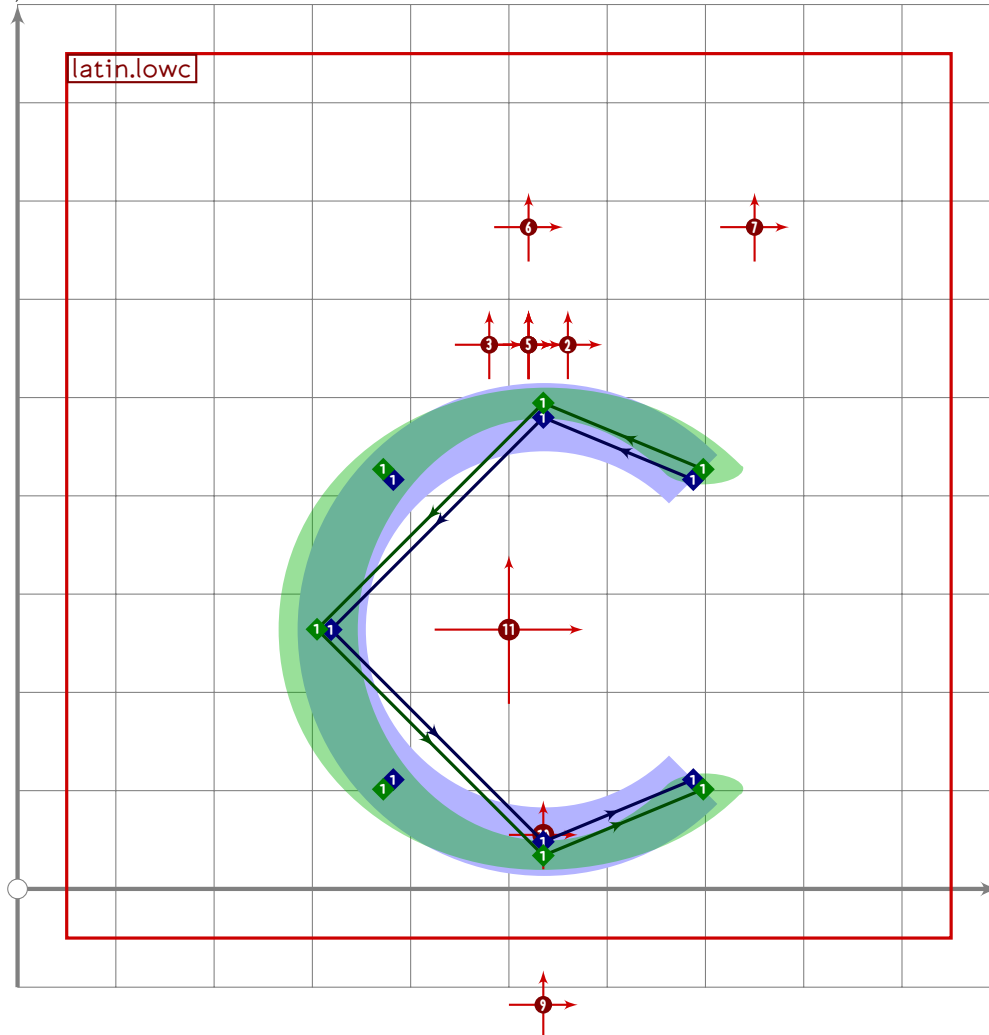
1021
1022 vardef latin.lowb =
1023   push_pbox_toexpand("latin.lowb");
1024   (x1+x4)/2=500;
1025   (x4-x1)=(y1-y3)*0.55;
1026   x2=x1=x6;
1027   x3=0.4[x2,x4];
1028   x5=0.55[x2,x4];
1029
1030   y1=latin_wide_high_v;
1031   y2=latin_wide_lc_baselift;
1032   y3=latin_wide_low_r;
1033   y4=0.48[y3,y5];
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;

```



```

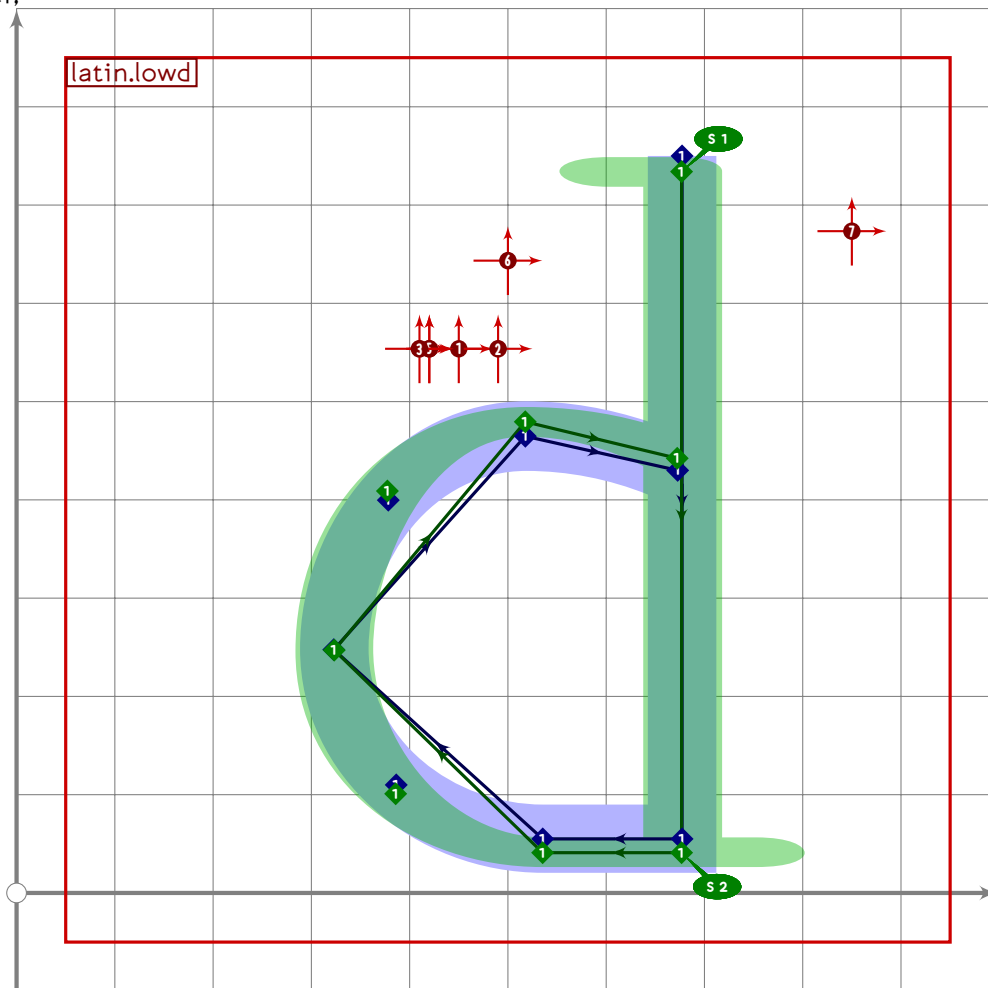
1050
1051 vardef latin.lowc =
1052   push_pbox_toexpand("latin.lowc");
1053   push_stroke((subpath (0.5,3.5) of ((1,0)..(0,1)..(-1,0)..(0,-1)..cycle))
1054     scaled ((latin_wide_xheight_r-latin_wide_low_r)/2)
1055     shifted ((xpart centre_pt,(latin_wide_xheight_r+latin_wide_low_r)/2)
1056       +(35,0)),
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)

```

```

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));

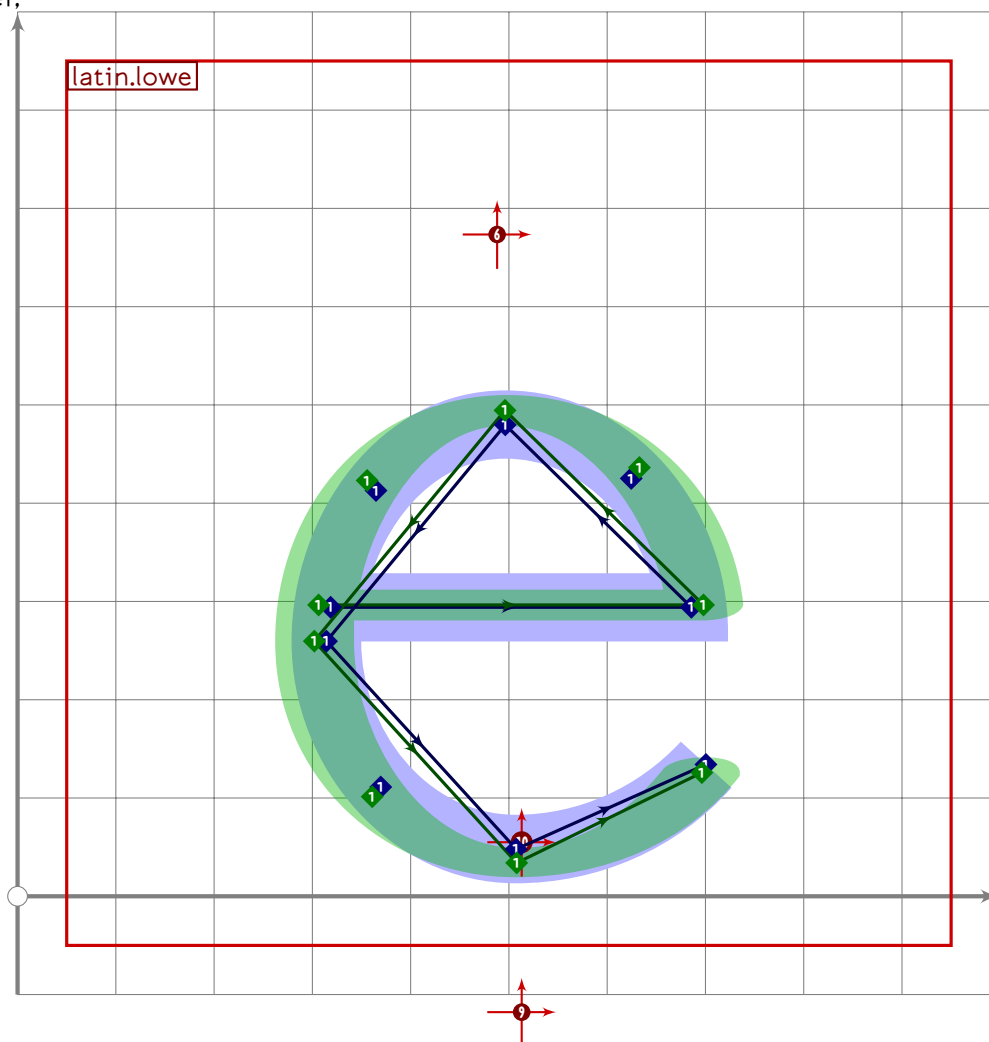
```


U+FF45
tsuku.uniFF45

```

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,accent_default[anc_caron_comma] shifted (120,0));
1096 expand_pbox;
1097 enddef;

```



LATI

```

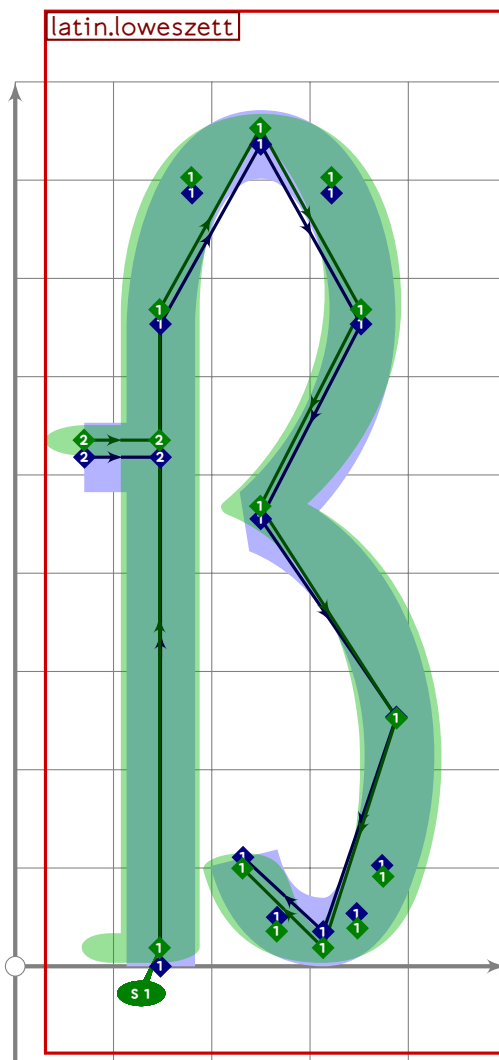
1098
1099 vardef latin.love =
1100   push_pbox_toexpand("latin.love");
1101   y2=0.57[y5,y3];
1102   y3=latin_wide_xheight_r;

```

```

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

```



```

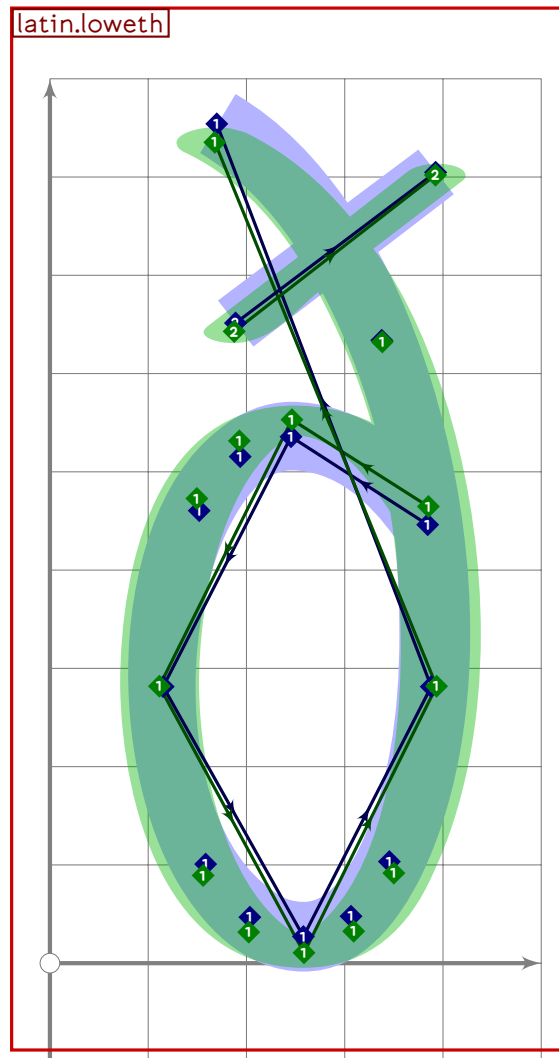
1123
1124 vardef latin.loweszett =
1125   push_pbox_toexpand("latin.loweszett");
1126   (x1+x6)/2=510;
1127   (x6-x1)=3*(y3-y2);
1128   x2=x1;
1129   x3=x5=(x1+x4)/2;
1130   x4=0.85[x1,x6];
1131   x7=0.69[x1,x6];
1132   x8=0.35[x1,x6];
1133
1134   y1=latin_wide_low_v;
1135   y2=y4=0.52[y5,y3];
1136   y3=latin_wide_high_r;
1137   y5=0.87[y7,latin_wide_xheight_h];
1138   y6=0.52[y7,y5];
1139   y7=latin_wide_low_h;
1140   y8=0.18[y7,y5];
1141

```

```

1142 push_stroke(z1-z2{dir 88}..z3{right}..z4.{dir 200}z5{dir 350}..
1143     z6..z7{left}..z8{dir 120},
1144     (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-
1145     (1.6,1.6)-(1.6,1.6)-(1.6,1.6));
1146 set_botip(0,4,0);
1147 set_boserif(0,0,1);
1148
1149 push_stroke((x1-150,latin_wide_xheight_h)-(x1,latin_wide_xheight_h),
1150     (1.6,1.6)-(1.6,1.6));
1151 expand_pbox;
1152 enddef;

```



```

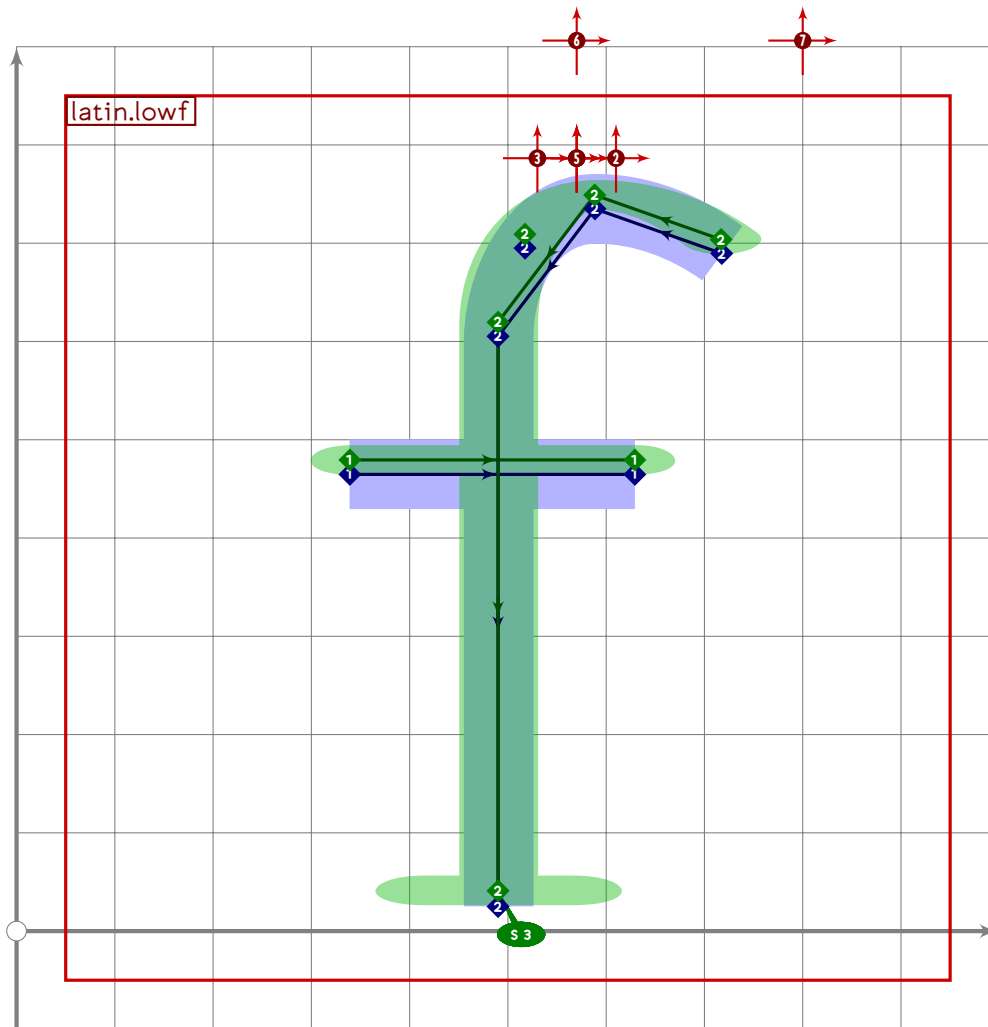
1153
1154 vardef latin.loweth =
1155     push_pbox_toexpand("latin.loweth");
1156     (x3+x5)/2=(x2+x4)/2=510;
1157     x4-x2=20;
1158     (x5-x3)=(y2-y4);
1159     x1=1.3[x3,x5];
1160     x6=0.2[x3,x5];

```

```

1161
1162 y1=0.25[y4,y2];
1163 y3=y5=0.5[y4,y2];
1164 y2=latin_wide_xheight_r;
1165 y4=latin_wide_low_r;
1166 y6=latin_wide_high_v;
1167
1168 push_stroke(z1..z2{left}..z3..z4{right}..z5.{curl 0.6}z6,
1169   (1.6,1.6)–(1.6,1.6)–(1.6,1.6)–(1.6,1.6)–(1.6,1.6)–(1.6,1.6));
1170 replace_strokep(0)(subpath (xpart ((subpath (0,1) of oldp)
1171   intersectiontimes
1172     (subpath (2,infinity) of oldp)),infinity) of oldp);
1173
1174 z7=point 4.67 of get_strokep(0);
1175 z8=z7+whatever*dir 202;
1176 x8=0.27[x3,x5];
1177 z9=2[z8,z7];
1178
1179 push_stroke(z8–z9,(1.5,1.5)–(1.5,1.5));
1180 set_bosize(0)(85);
1181 expand_pbox;
1182 endif;

```



```

1183
1184 vardef latin.lowf =
1185   push_pbox_toexpand("latin.lowf");
1186   (x2-x1)=290;
1187   x5=x6=490=0.52[x1,x2];
1188   x3-x5=2*(y4-y5);
1189   x4=0.38[x5,x3];
1190
1191   y1=y2=latin_wide_xheight_h;
1192   y5=0.52[y2,y4];
1193   y3=0.73[y2,y4];
1194   y4=latin_wide_high_r;
1195   y6=latin_wide_low_v;
1196
1197   push_stroke(z1-z2,(1.6,1.6)-(1.6,1.6));
1198
1199   push_stroke(z3{curl 0.6}..z4{left}..{dir 268}z5{down}-z6,
1200     (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
1201   replace_strokep(0)(subpath (0.23,3) of oldp);
1202   set_boserif(0,3,3);

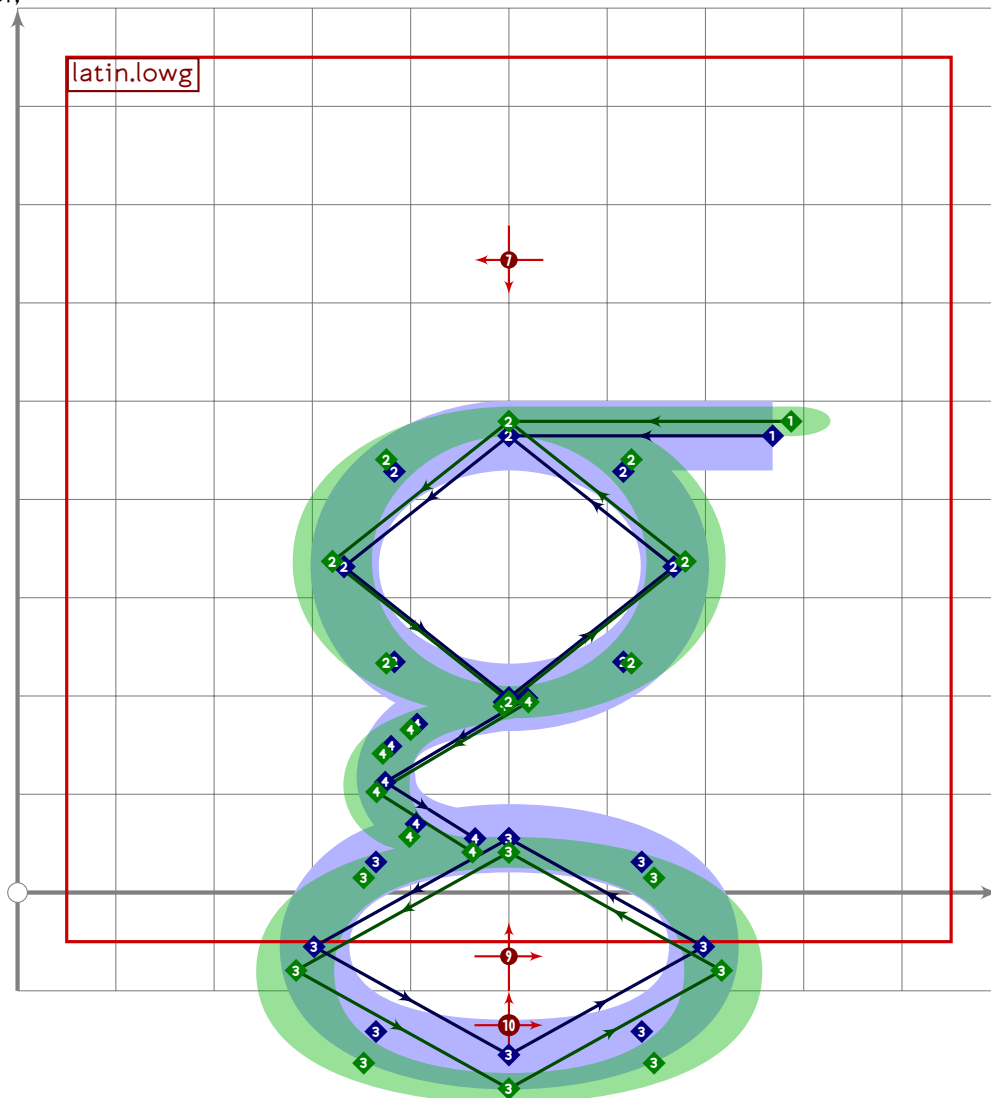
```

U+FF47
tsuku.uniFF47

```

1203
1204 tsu_accent.shift_anchors(ypart olda>vmetric(0.52))
1205 (((0,0) transformed tsu_xf.cap_upper_accent)-
1206 ((0,0) transformed accent_default[anc_upper])+(70,0));
1207 expand_pbox;
1208 enddef;

```



```

1209
1210 vardef latin.lowg =
1211   push_pbox_toexpand("latin.lowg");
1212   x2=x4=x7=x9=500;
1213   x1=1.6[x2,x5];
1214   x5-x2=x2-x3;
1215   x5-x3=1.26*(y2-y4);
1216   x6=0.25[x3,x2];
1217   x10-x7=x7-x8;
1218   x10-x8=1.8*(y7-y9);
1219
1220   y1=y2=latin_wide_xheight_h;

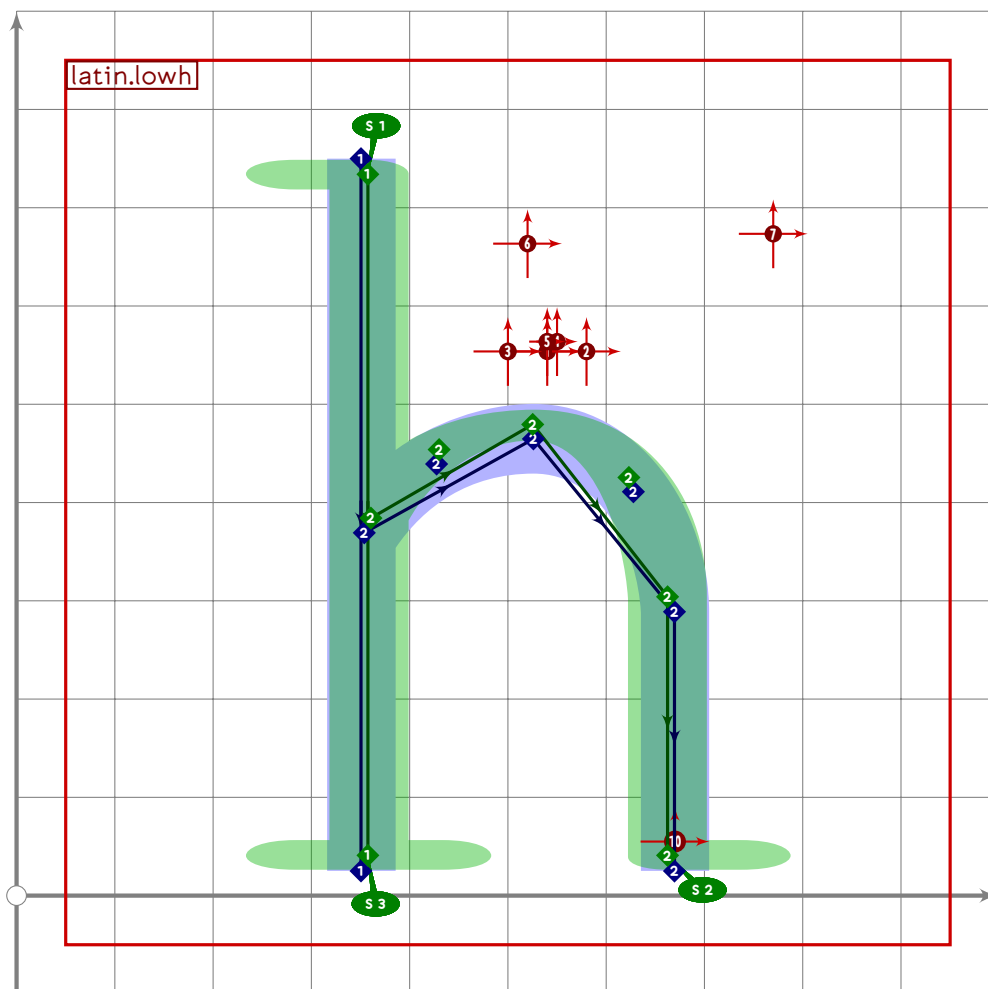
```

LATI

```

1221 y3=y5=0.5[y4,y2];
1222 y4=0.35[y7,y2];
1223 y6=0.4[y7,y4];
1224 y7=latin_wide_low_h;
1225 y8=y10=0.5[y9,y7];
1226 y9=latin_wide_desc_r;
1227
1228 push_stroke(z1-z2,(1.6,1.6)-(1.6,1.6));
1229
1230 push_stroke(z3..z4..z5..z2..cycle,
1231   (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-cycle);
1232
1233 push_stroke(z7..z8..z9..z10..cycle,
1234   (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-cycle);
1235
1236 push_stroke((point 1.2 of get_strokep(-1))
1237   {-direction 1.2 of get_strokep(-1)}..z6..
1238   (point 0.05 of get_strokep(0)) {-direction 0.05 of get_strokep(0)},
1239   (1.6,1.6)-(1.6,1.6)-(1.6,1.6));
1240 replace_strokep(0)(subpath (0.13,1.87) of oldp);
1241 set_bosize(0,85);
1242
1243 push_anchor(anc_caron_comma,
1244   identity rotated 180 shifted (0,90)
1245   transformed accent_default[anc_upper]);
1246 push_anchor(anc_lower,
1247   accent_default[anc_lower] shifted (0,53));
1248 push_anchor(anc_lower_connect,
1249   accent_default[anc_lower_connect] shifted (0,190));
1250 expand_pbox;
1251 endif;

```

```

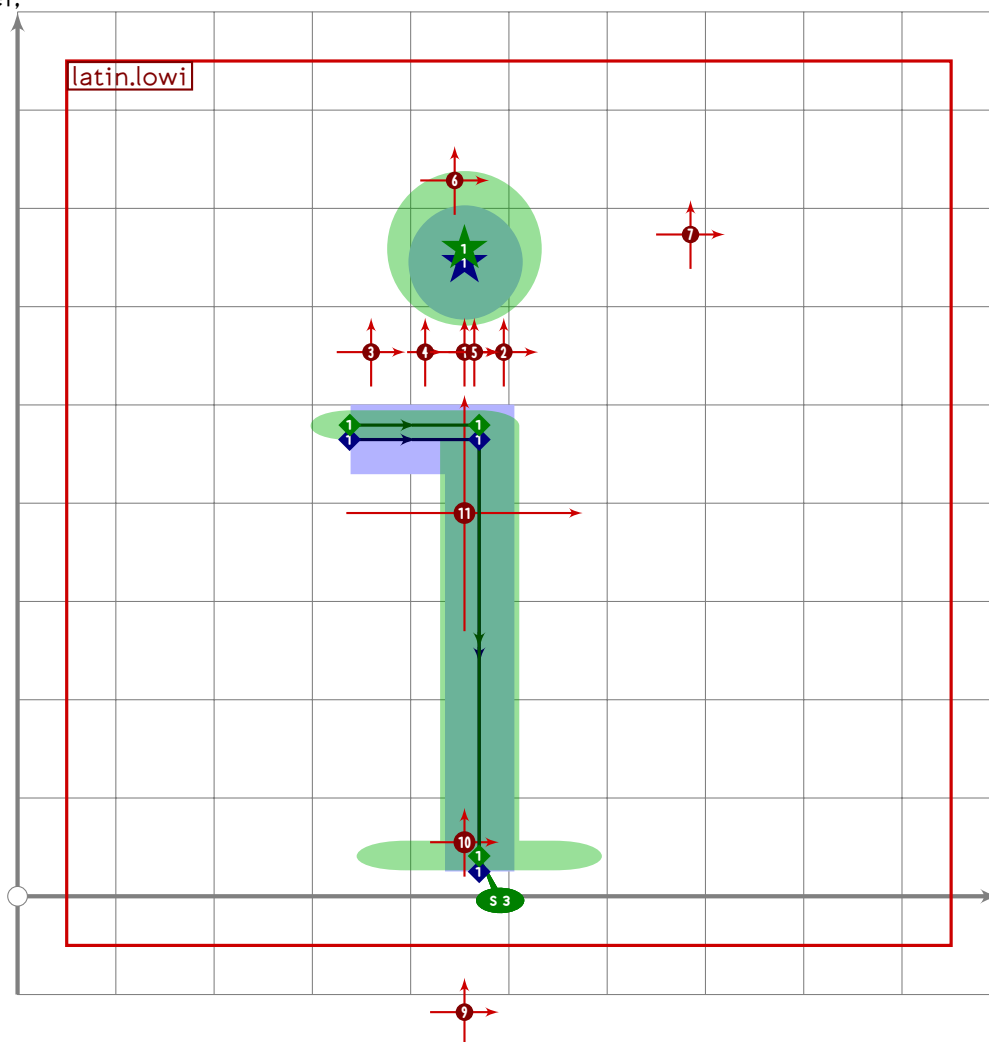
1252
1253 vardef latin.lowh =
1254   push_pbox_toexpand("latin.lowh");
1255   (x1+x5)/2=510;
1256   (x5-x1)=(y1-y2)*0.44;
1257   x2=x1=x3;
1258   x4=0.55[x3,x5];
1259   x6=x5;
1260
1261   y1=latin_wide_high_v;
1262   y2=y6=latin_wide_low_v;
1263   y3=0.77[y2,y4];
1264   y4=latin_wide_xheight_h;
1265   y5=0.60[y2,y4];
1266
1267   push_stroke(z1-z2,(1.6,1.6)-(1.6,1.6));
1268   if not do_italic_hook:
1269     set_boserif(0,0,1);
1270     set_boserif(0,1,3);
1271   fi;
1272

```

```

1273 push_stroke(z3..z4{right}..z5{dir 273}-z6,
1274   (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
1275 replace_stroke(0)(subpath (0.03,3) of oldp);
1276 set_boserif(0,3;if do_italic_hook: 11 else: 2 fi);
1277
1278 tsu_accent.shift_anchors(ypart olda>vmetric(0.52))((40,0));
1279 push_anchor(anc_wide,identity xscaled 0.8
1280   transformed accent_default[anc_wide] shifted (50,10));
1281 push_anchor(anc_tilde,identity xscaled 0.7
1282   transformed accent_default[anc_tilde] shifted (40,10));
1283 push_anchor(anc_ring,accent_default[anc_ring] shifted (20,-10));
1284 push_anchor(anc_lower_connect,
1285   accent_default[anc_lower_connect] shifted (170,0));
1286 expand_pbox;
1287 enddef;

```



```

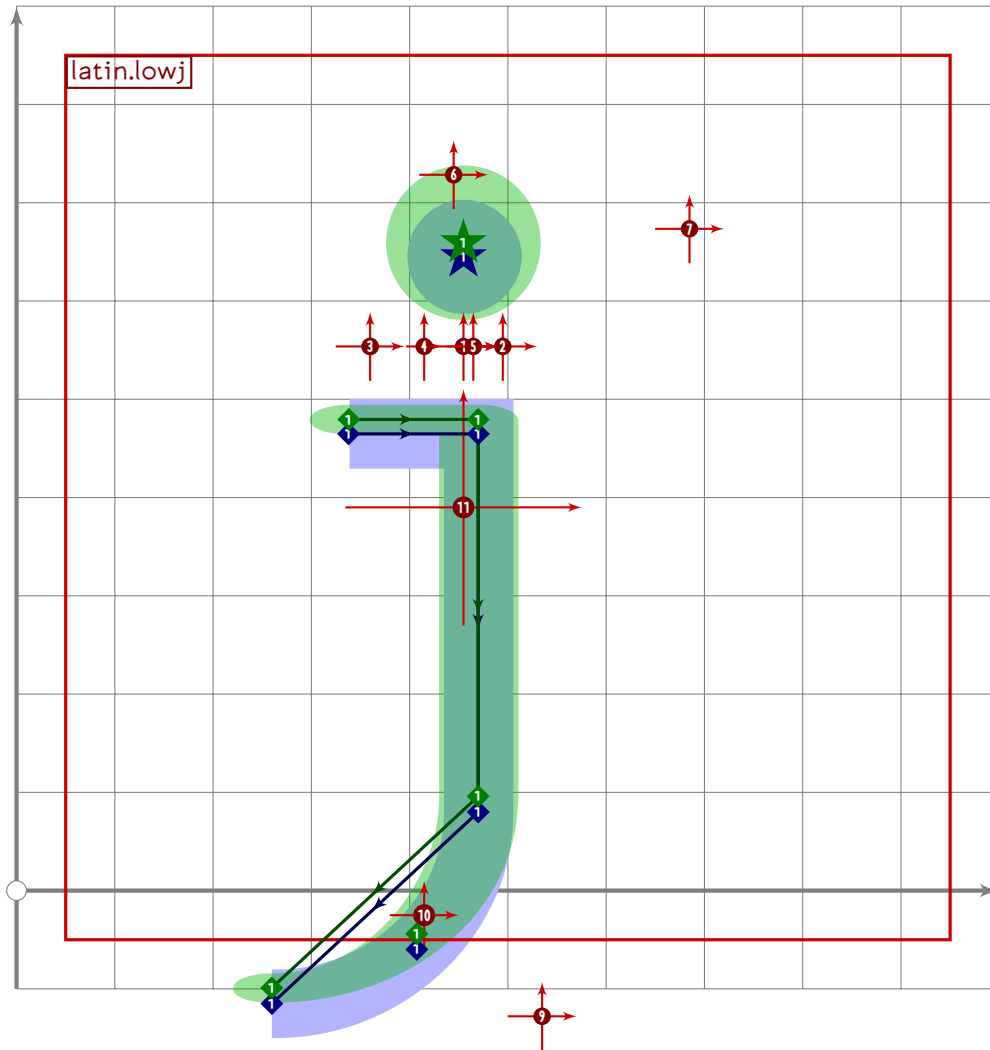
1288
1289 vardef latin.lowi =
1290   push_pbox_toexpand("latin.lowi");
1291   x2=x3;
1292   0.85[x1,x2]=450;

```

```

1293 (x2-x1)=0.3(y2-y3);
1294 x4=x2-15;
1295
1296 y1=y2=latin_wide_xheight_h;
1297 y3=latin_wide_low_v;
1298 y4=0.5[y2,latin_wide_high_v]+mbrush_width;
1299
1300 push_stroke(z1-z2-z3,(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
1301 set_botip(0,1,1);
1302 set_boserif(0,2,3);
1303
1304 push_lcblob(fullcircle rotated 45 scaled (mbrush_width*2.7+15)
1305   shifted (z4 transformed tsu_rescale_xform)
1306   transformed inverse tsu_rescale_xform);
1307
1308 push_anchor(anc_wide,
1309   identity xscaled 0.7 transformed accent_default[anc_wide]);
1310 tsu_accent.shift_anchors(true)((x4-500,0));
1311 tsu_accent.shift_anchors(ai=anc_acute)((-55,0));
1312 tsu_accent.shift_anchors(ai=anc_wide)((-40,0));
1313 tsu_accent.shift_anchors(ai=anc_tilde)((10,0));
1314 tsu_accent.shift_anchors(ai=anc_ring)((-10,55));
1315 expand_pbox;
1316 endif;

```



```

1317
1318 vardef latin.lowj =
1319   push_pbox_toexpand("latin.lowj");
1320   x2=x3;
1321   0.85[x1,x2]=450;
1322   (x2-x1)=0.3(y2-y3);
1323   x5=x2-15;
1324
1325   y1=y2=latin_wide_xheight_h;
1326   y3=latin_wide_low_v;
1327   y5=0.5[y2,latin_wide_high_v]+mbrush_width;
1328
1329   z4=z3+(-210,-140);
1330
1331   push_stroke((z1-z2-(z3+(0,55)))..{curl 0.8}z4,
1332     (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
1333   set_botip(0,1,1);
1334
1335   push_lcblob(fullcircle scaled (mbrush_width*2.7*15)

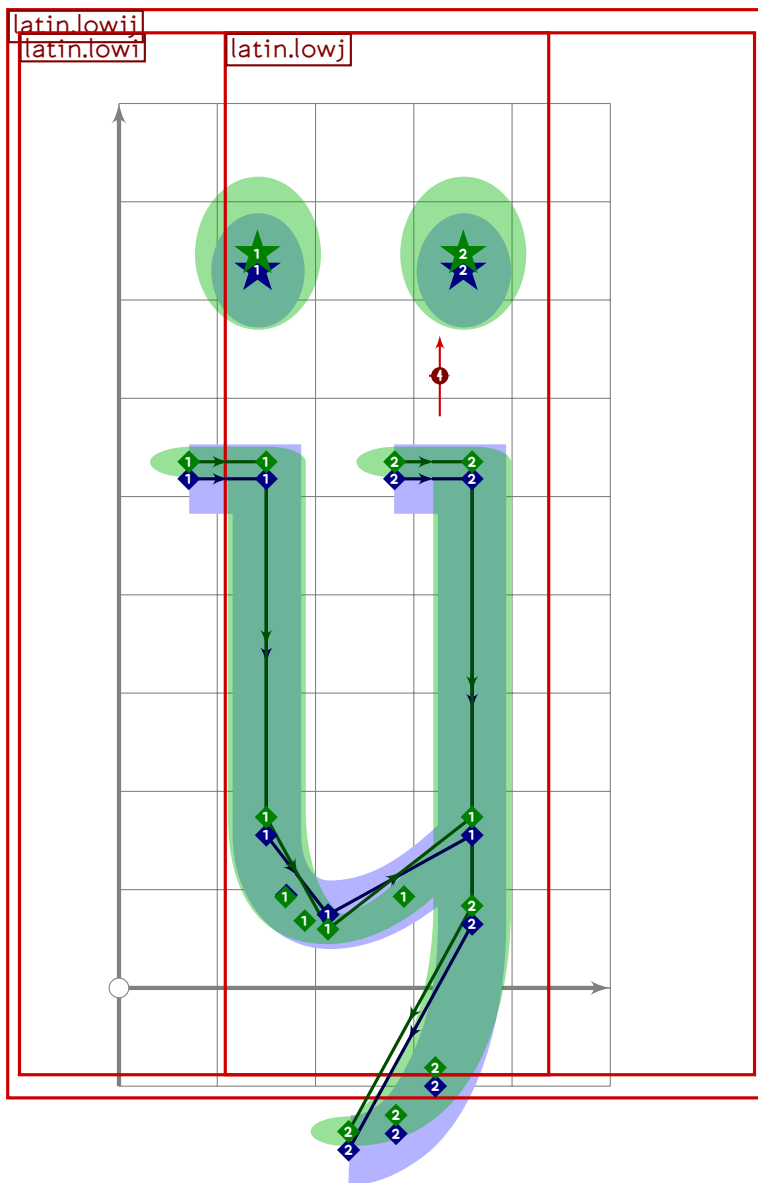
```

U+0133
tsuku.ij

```

1336 shifted (z5 transformed tsu_rescale_xform)
1337 transformed inverse tsu_rescale_xform);
1338
1339 push_anchor(anc_wide,
1340 identity xscaled 0.7 transformed accent_default[anc_wide]);
1341 tsu_accent.shift_anchors(true)((x5-500,0));
1342 tsu_accent.shift_anchors(ai=anc_acute)((-55,0));
1343 tsu_accent.shift_anchors(ai=anc_wide)((-40,0));
1344 tsu_accent.shift_anchors(ai=anc_tilde)((10,0));
1345 tsu_accent.shift_anchors(ai=anc_ring)((-10,55));
1346 tsu_accent.shift_anchors(ai=anc_lower)((80,-10));
1347 tsu_accent.shift_anchors(ai=anc_lower_connect)((-40,80));
1348 expand_pbox;
1349 endif;

```



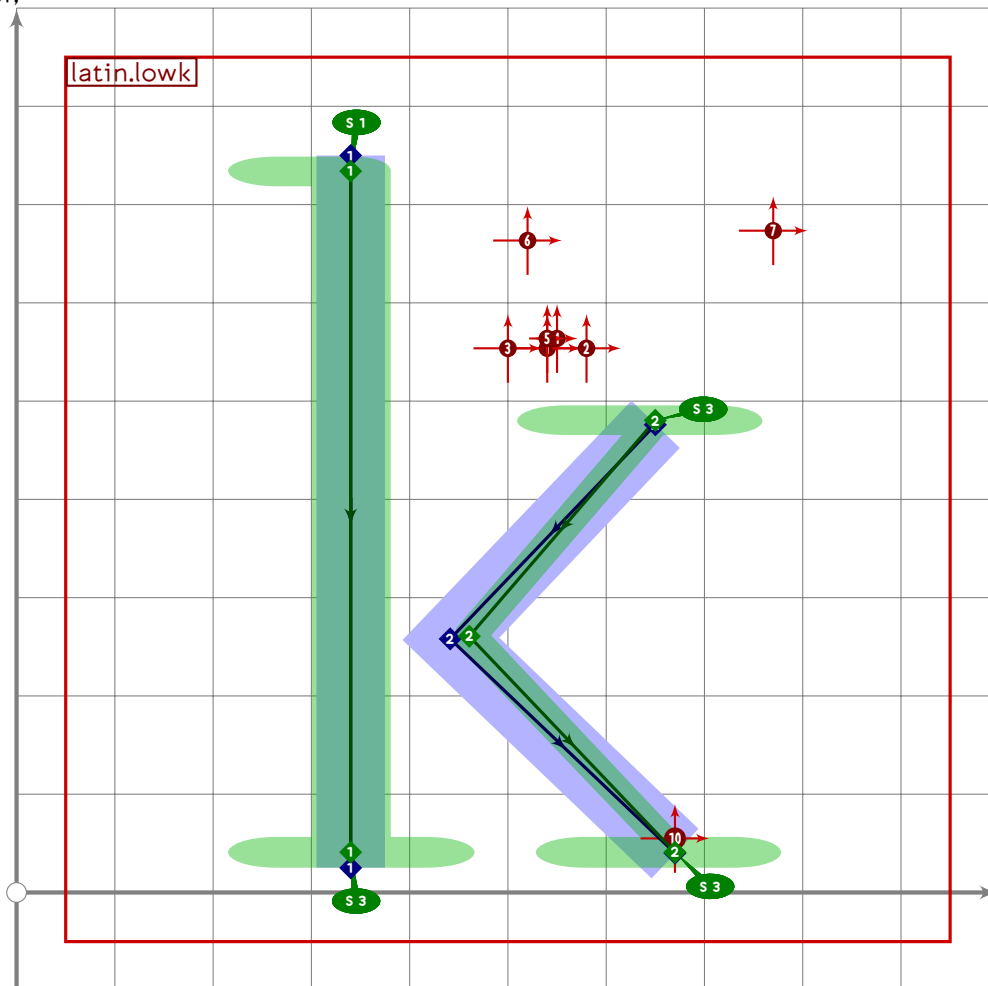
LATI

```
1350
1351 vardef latin.lowij =
```

```

1352 push_pbox_toexpand("latin.lowij");
1353 tsu_xform(identity shifted (-200,0))(latin.lowi);
1354 numeric x[],y[];
1355 tsu_xform(identity shifted (150,0))(latin.lowj);
1356 replace_lcblob(-1)(get_lcblob(0) shifted (-350,0));
1357 numeric x[],y[];
1358 z1=point 1.7 of get_stroke(-1);
1359 x2=0.3[x1,x3];
1360 y2=vmetric(0.05);
1361 z3=point 1.8 of get_stroke(0);
1362 replace_stroke(-1)((subpath (0,1) of oldp)-
1363   z1{dir 273}..z2{right}..{curl 0.3}z3);
1364 replace_strokeq(-1)((subpath (0,1) of oldq)-(1.6,1.6)-(1.6,1.6)-(1,1));
1365 set_boserif(-1,2,whatever);
1366 expand_pbox;
1367 enddef;

```



```

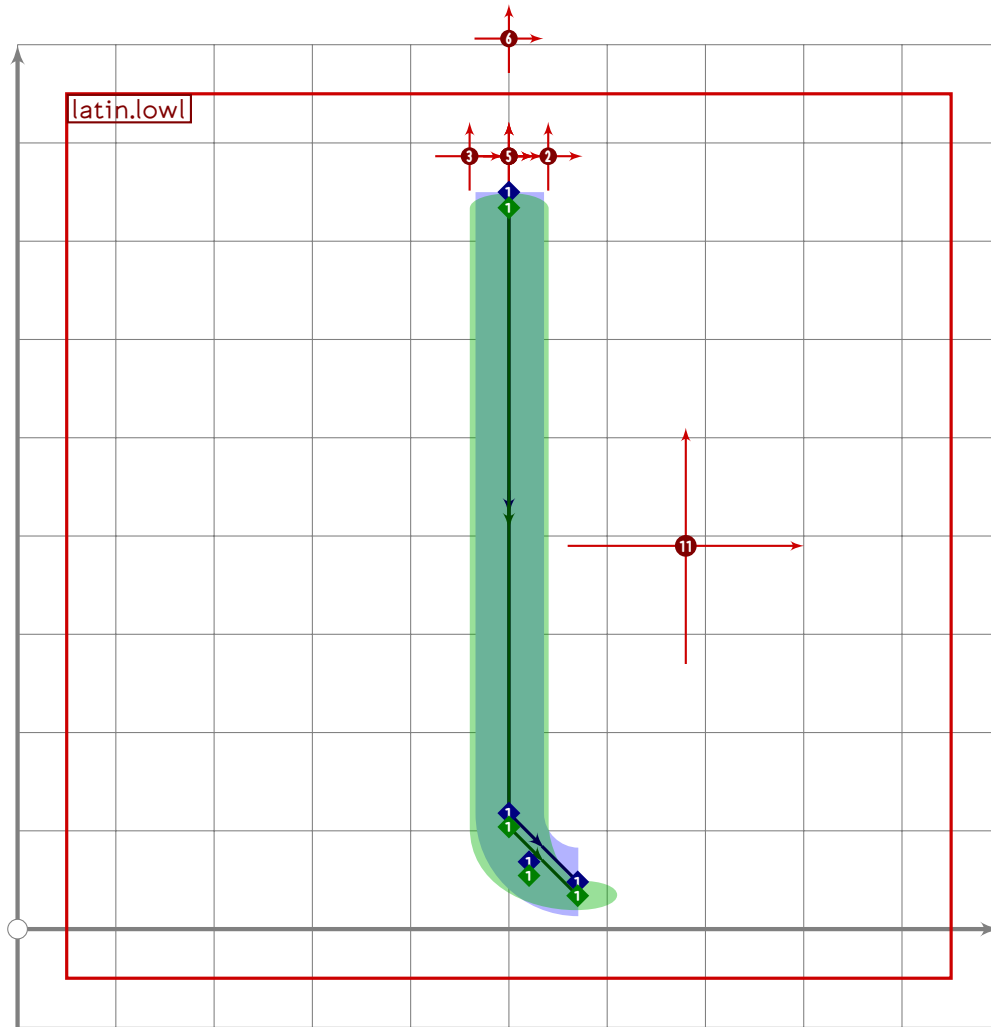
1368
1369 vardef latin.lowk =
1370   push_pbox_toexpand("latin.lowk");
1371   z1=(340,latin_wide_high_v);
1372   z2=(340,latin_wide_low_v);

```

```

1373 z3=(650,(latin_wide_xheight_v+latin_wide_xheight_h)/2);
1374 x4=340+mbrush_width*if sharp_corners: 2.7 else: 2.3 fi;
1375 y4=(y3+y5)/2;
1376 z5=(670,0.5[latin_wide_low_h,latin_wide_low_v]);
1377
1378 push_stroke(z1-z2,(1.6,1.6)-(1.6,1.6));
1379 if not do_italic_hook:
1380     set_boserif(0,0,1);
1381     set_boserif(0,1,3);
1382 fi;
1383
1384 push_stroke(z3-z4-z5,(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
1385 set_botip(0,1,1);
1386 set_boserif(0,0,if do_italic_hook: 1 else: 3 fi);
1387 if not do_italic_hook: set_boserif(0,2,3); fi;
1388 set_bobrush(0,bralternate);
1389
1390 tsu_accent.shift_anchors(ypart olda>vmetric(0.52))((40,0));
1391 push_anchor(anc_wide,identity xscaled 0.8
1392     transformed accent_default[anc_wide] shifted (50,10));
1393 push_anchor(anc_tilde,identity xscaled 0.7
1394     transformed accent_default[anc_tilde] shifted (40,10));
1395 push_anchor(anc_ring,accent_default[anc_ring] shifted (20,-10));
1396 push_anchor(anc_lower_connect,
1397     accent_default[anc_lower_connect] shifted (170,0));
1398 expand_pbox;
1399 enddef;

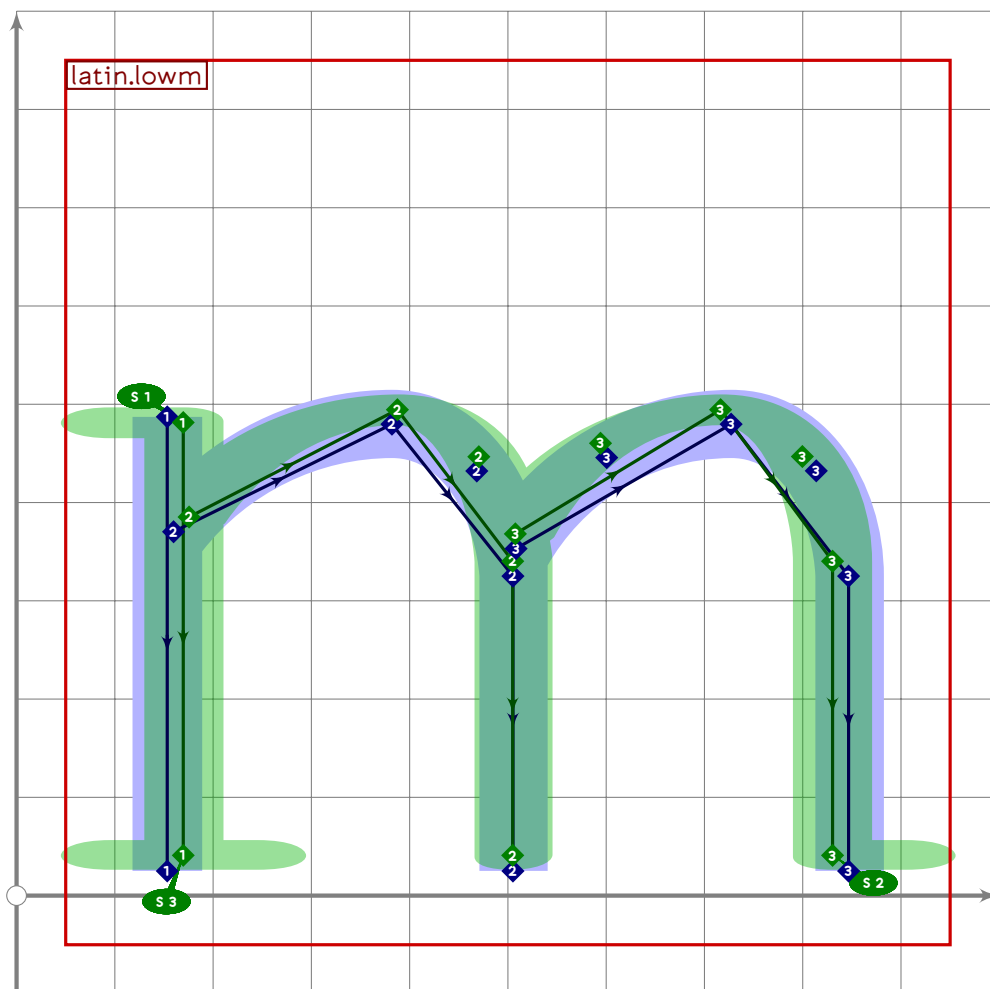
```



```

1400
1401 vardef latin.lowl =
1402   push_pbox_toexpand("latin.lowl");
1403   x1=x2=500;
1404   x3=570;
1405
1406   y1=latin_wide_high_v;
1407   y2=y3+(x3-x2);
1408   y3=latin_wide_low_r;
1409
1410   push_stroke(z1-z2{down}..{right}z3,(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
1411
1412   tsu_accent.shift_anchors((ypart olda>vmetric(0.52))
1413                           and not (ai=anc_caron_comma))
1414   (((0,0) transformed tsu_xf.cap_upper_accent)-
1415    ((0,0) transformed accent_default[anc_upper]));
1416   tsu_accent.shift_anchors(ai=anc_centre)((180,0));
1417   expand_pbox;
1418 enddef;

```

```

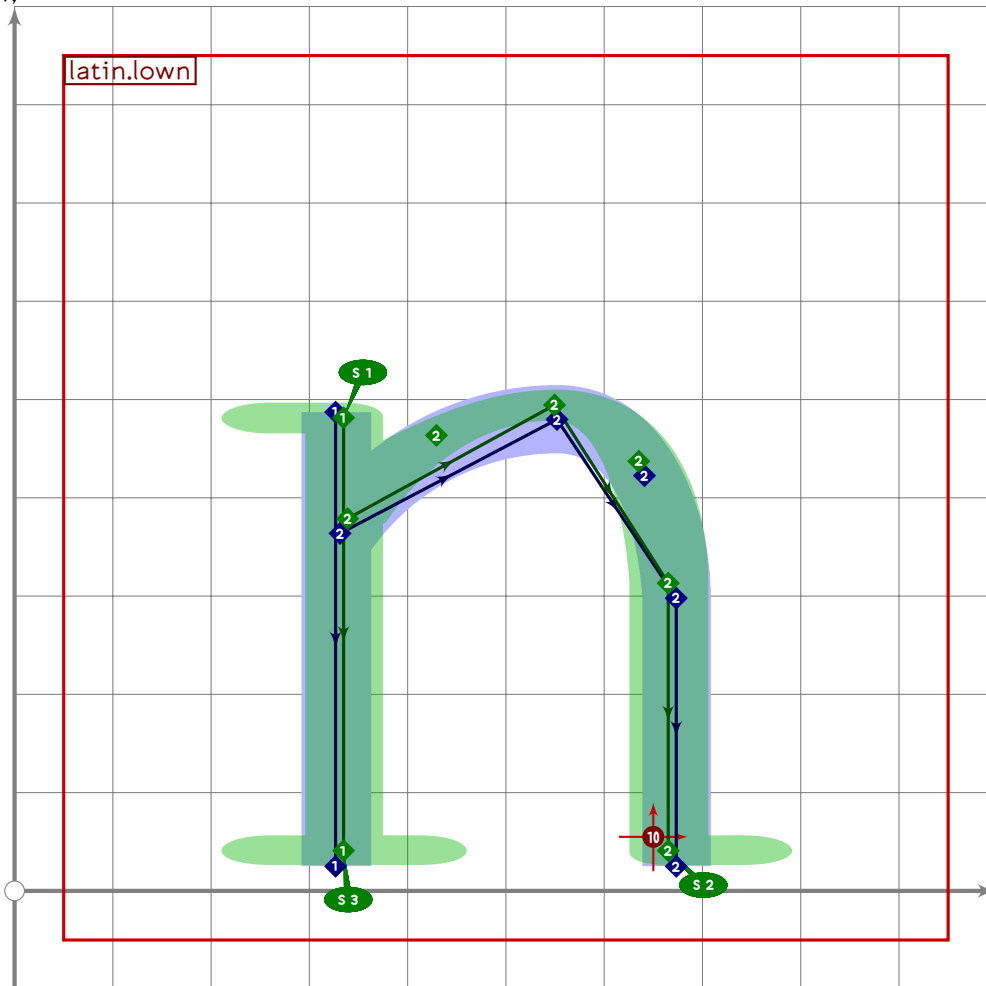
1419
1420 vardef latin.lowm =
1421   push_pbox_toexpand("latin.lowm");
1422   (x1+x9)/2=500;
1423   (x9-x1)*2=(y1-y2)*3;
1424   (x5-x1)=(x9-x5)*1.03;
1425   x2=x1=x3;
1426   x4=0.65[x3,x5];
1427   x6=x5;
1428   x8=0.65[x6,x9];
1429   x9=x10;
1430
1431   y1=latin_wide_xheight_v;
1432   y2=y6=y10=latin_wide_low_v;
1433   y3=0.74[y2,y4];
1434   y4=y8=latin_wide_xheight_r;
1435   y5=y9=0.66[y2,y4];
1436
1437   push_stroke(z1-z2,(1.6,1.6)-(1.6,1.6));
1438   set_boserif(0,0;if do_italic_hook: 11 else: 1 fi);
1439   if not do_italic_hook: set_boserif(0,1,3); fi;

```

```

1440
1441 push_stroke(z3..z4{right}..z5{dir 275}-z6,
1442   (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
1443 replace_stroke(0)(subpath (0.04,3) of oldp);
1444
1445 z7=get_stroke(0) intersectionpoint (z3-z9);
1446
1447 push_stroke(z7..z8{right}..z9{dir 271}-z10,
1448   (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
1449 replace_stroke(0)(subpath (0.04,3) of oldp);
1450 set_boserif(0,3;if do_italic_hook: 11 else: 2 fi);
1451 expand_pbox;
1452 enddef;

```



```

1453
1454 vardef latin.lown =
1455   push_pbox_toexpand("latin.lown");
1456   (x1+x5)/2=500;
1457   (x5-x1)=(y1-y2)*0.75;
1458   x2=x1+x3;
1459   x4=0.65[x3,x5];
1460   x6=x5;

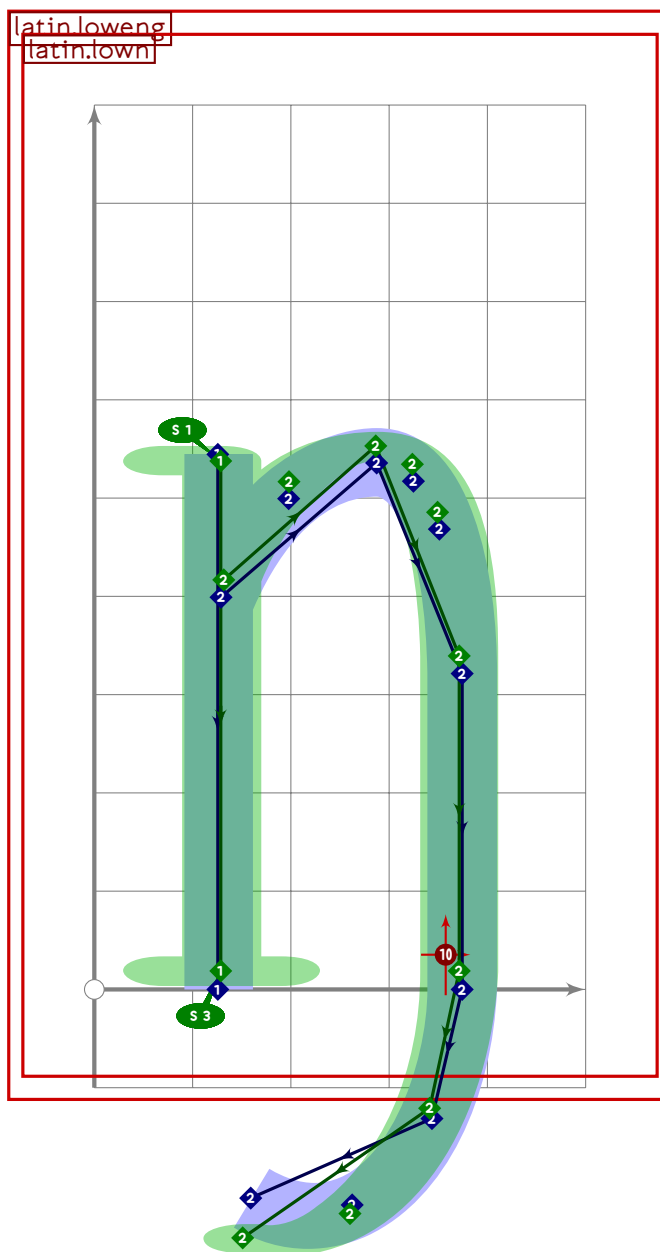
```

LATI

```

1461
1462 y1=latin_wide_xheight_v;
1463 y2=y6=latin_wide_low_v;
1464 y3=0.73[y2,y4];
1465 y4=latin_wide_xheight_r;
1466 y5=0.60[y2,y4];
1467
1468 push_stroke(z1-z2,(1.6,1.6)-(1.6,1.6));
1469 set_boserif(0,0;if do_italic_hook: 11 else: 1 fi);
1470 if not do_italic_hook: set_boserif(0,1,3); fi;
1471
1472 push_stroke(z3..z4{right}..z5{dir 273}-z6,
1473   (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
1474 replace_strokep(0)(subpath (0.03,3) of oldp);
1475 set_boserif(0,3;if do_italic_hook: 11 else: 2 fi);
1476
1477 push_anchor(anc_lower_connect,
1478   accent_default[anc_lower_connect] shifted (150,0));
1479 expand_pbox;
1480 enddef;

```

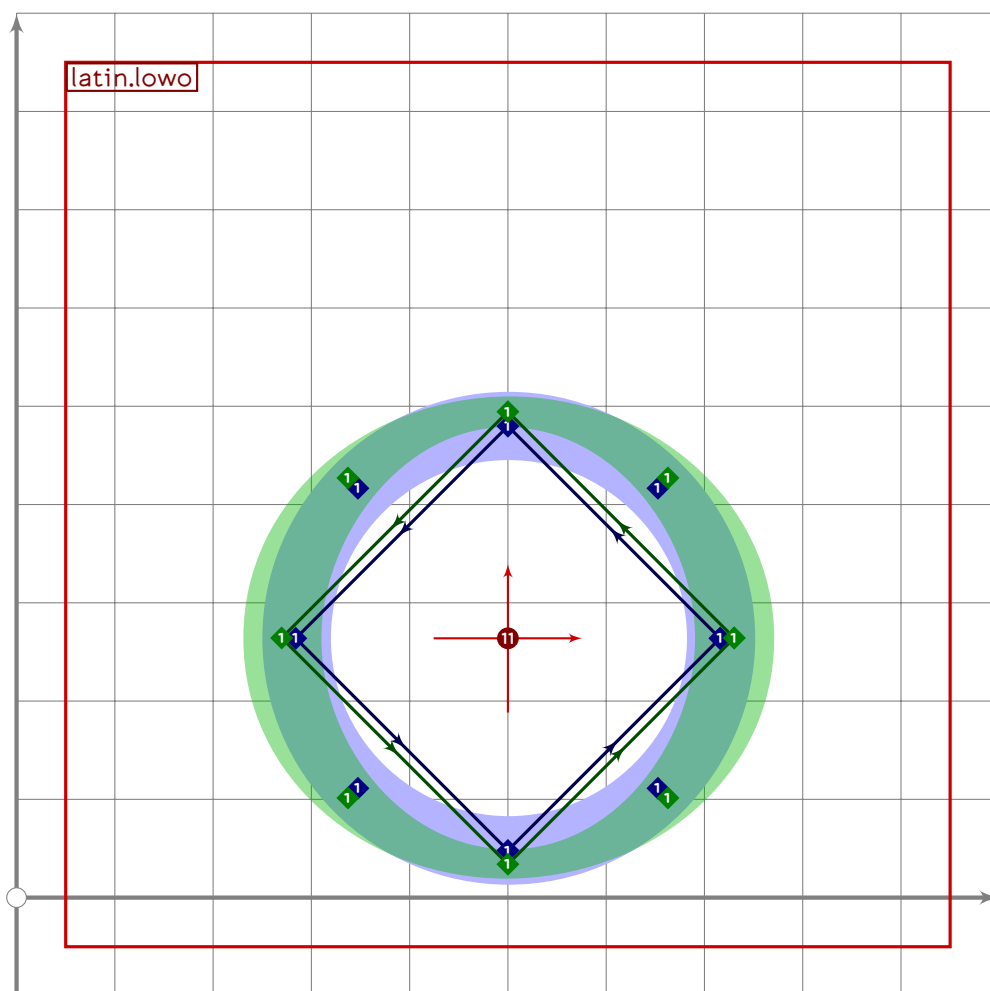


```

1481
1482 vardef latin.loweng =
1483   push_pbox_toexpand("latin.loweng");
1484   latin.lown;
1485   x7=x6-300;
1486   y7=latin_wide_desc_h;
1487   replace_strokep(0)(oldp{dir 266}..{curl 0.8}z7);
1488   replace_strokep(0)(insert_nodes(oldp)(3.3));
1489   replace_strokeq(0)(oldq-(1.6,1.6)-(1.6,1.6));
1490   set_boserif(0,3,whatever);
1491   expand_pbox;
1492 enddef;

```

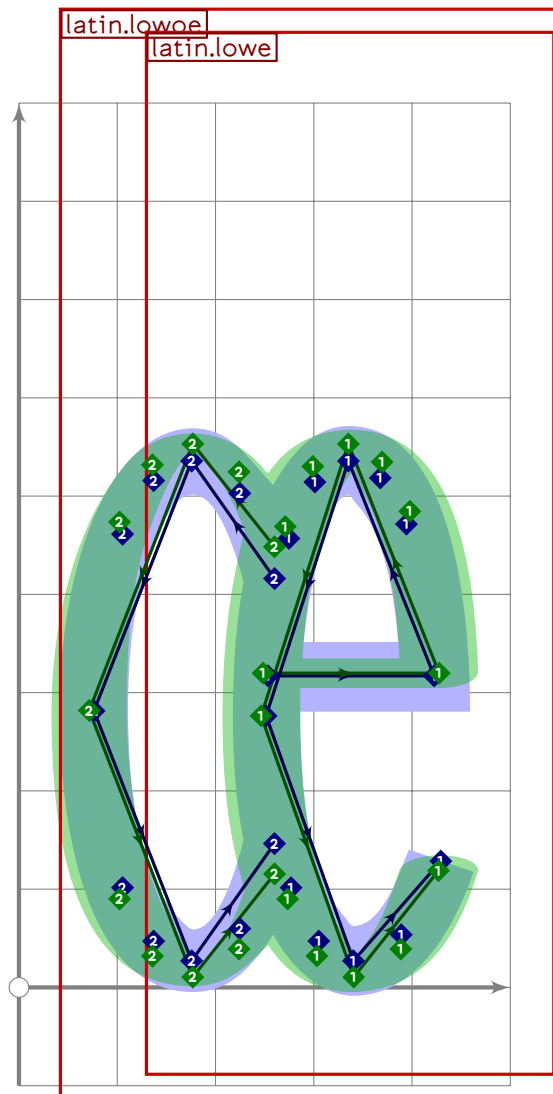
U+FF4F
tsuku.uniFF4F



```

1493
1494 vardef latin.lowo =
1495   push_pbox_toexpand("latin.lowo");
1496   push_anchor(anc_centre,identity
1497     scaled ((latin_wide_xheight_r-latin_wide_low_r)/200)
1498     shifted (xpart centre_pt,(latin_wide_xheight_r+latin_wide_low_r)/2));
1499   push_stroke(((1,0)..(0,1)..(-1,0)..(0,-1)..cycle)
1500     scaled ((latin_wide_xheight_r-latin_wide_low_r)/2)
1501     shifted (xpart centre_pt,(latin_wide_xheight_r+latin_wide_low_r)/2),
1502     (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-cycle);
1503   expand_pbox;
1504 enddef;

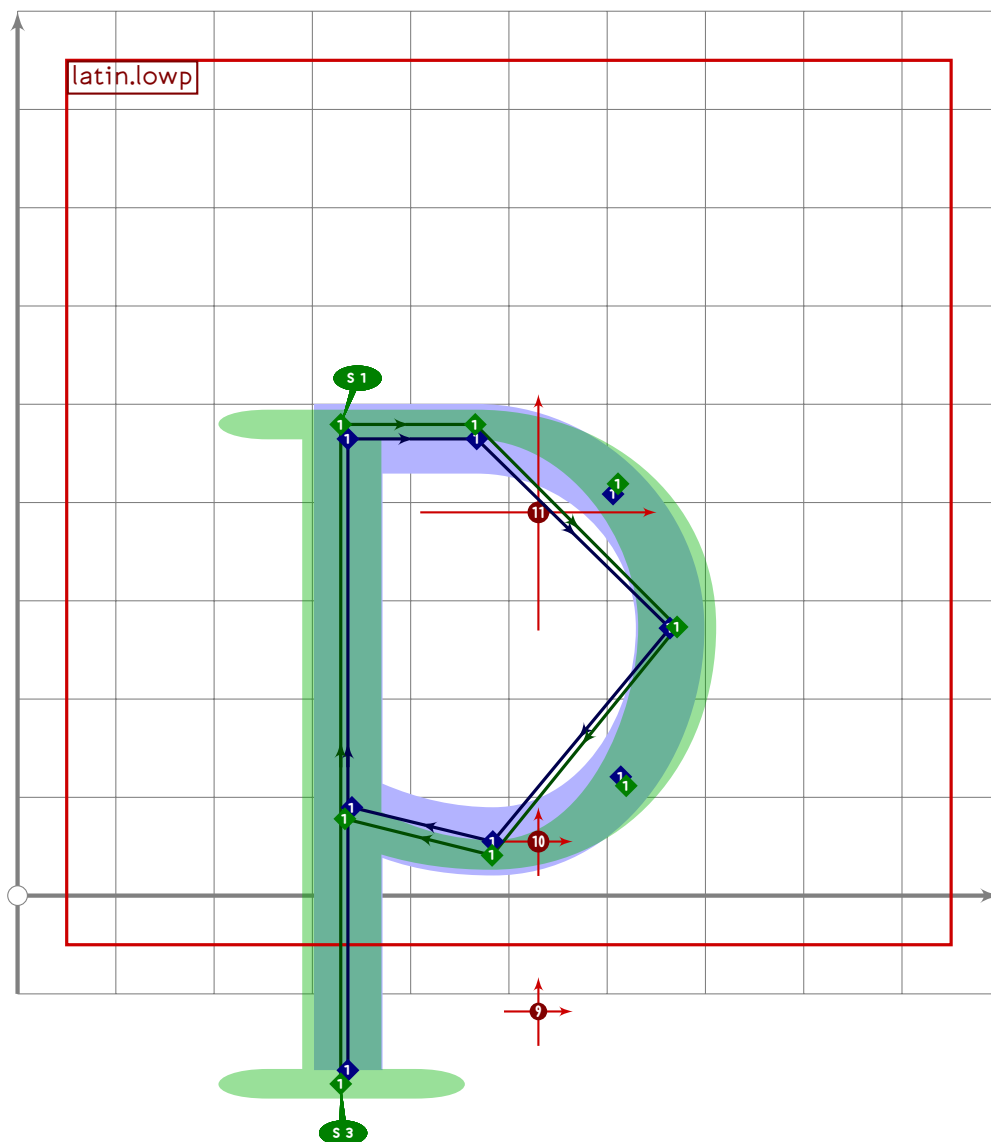
```



```

1505
1506 vardef latin.lowoe =
1507   push_pbox_toexpand("latin.lowoe");
1508   tsu_xform(identity shifted (190,0))(latin.lowe);
1509   push_stroke(((1,0)..(0,1)..(-1,0)..(0,-1)..(1,0))
1510     scaled ((latin_wide_xheight_r-latin_wide_low_r)/2)
1511     shifted (340,0.5[latin_wide_xheight_r,latin_wide_low_r],
1512       (1.2,1.2)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.2,1.2));
1513   replace_strokep(0)(subpath (0.03+xpart (oldp intersectiontimes
1514     (subpath (2,infinity) of get_strokep(-1))),
1515     3.97-xpart ((reverse oldp) intersectiontimes
1516       reverse get_strokep(-1)))
1517     of oldp);
1518   expand_pbox;
1519 enddef;

```



```

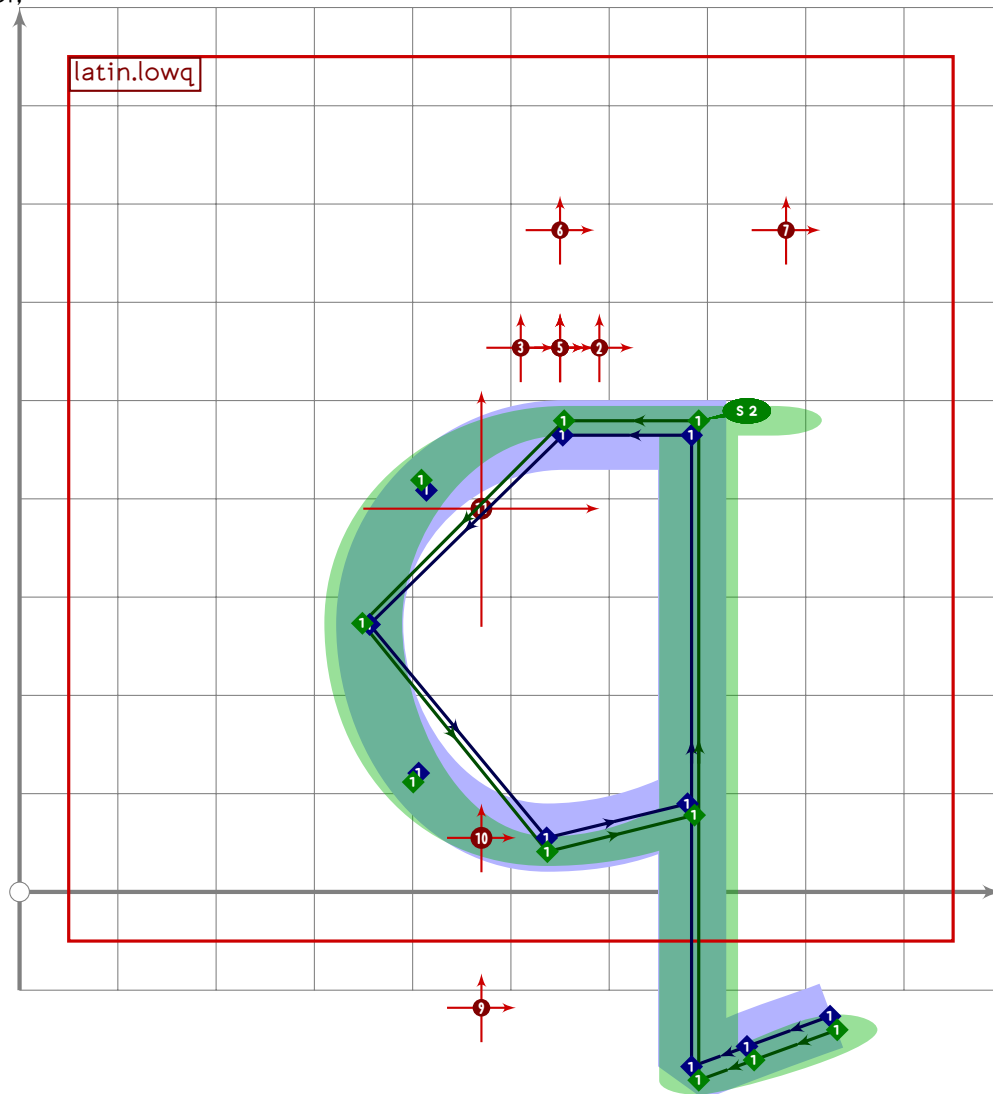
1520
1521 vardef latin.lowp =
1522   push_pbox_toexpand("latin.lowp");
1523   (x1+x4)/2=500;
1524   (x4-x1)=(y2-y1)*0.51;
1525   x2=x1=x6;
1526   x3=0.4[x2,x4];
1527   x5=0.45[x2,x4];
1528
1529   y1=latin_wide_desc_v;
1530   y2=y3=latin_wide_xheight_h;
1531   y4=0.47[y3,y5];
1532   y5=latin_wide_low_h;
1533   y6=0.91[y3,y5];
1534
1535   push_stroke(z1-z2{right}..{right}z3..{down}z4..{left}z5..z6,
1536     (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-
1537     (1.6,1.6)-(1.6,1.6));

```

```

1538 replace_strokep(0)(subpath (0,4.97) of oldp);
1539 set_botip(0,1,1);
1540 if not do_italic_hook: set_boserif(0,0,3); fi;
1541 set_boserif(0,1,1);
1542–1543
1544 tsu_accent.shift_anchors(ypart olda<vmetric(0.52))((30,0));
1545 expand_pbox;
1546 enddef;

```



```

1547
1548 vardef latin.lowq =
1549   push_pbox_toexpand("latin.lowq");
1550   (x1+x4)/2=520;
1551   (x1-x4)=(y2-y1)*0.51;
1552   x2=x1=x6;
1553   x3=0.4[x2,x4];
1554   x5=0.45[x2,x4];
1555
1556   y1=latin_wide_desc_v;

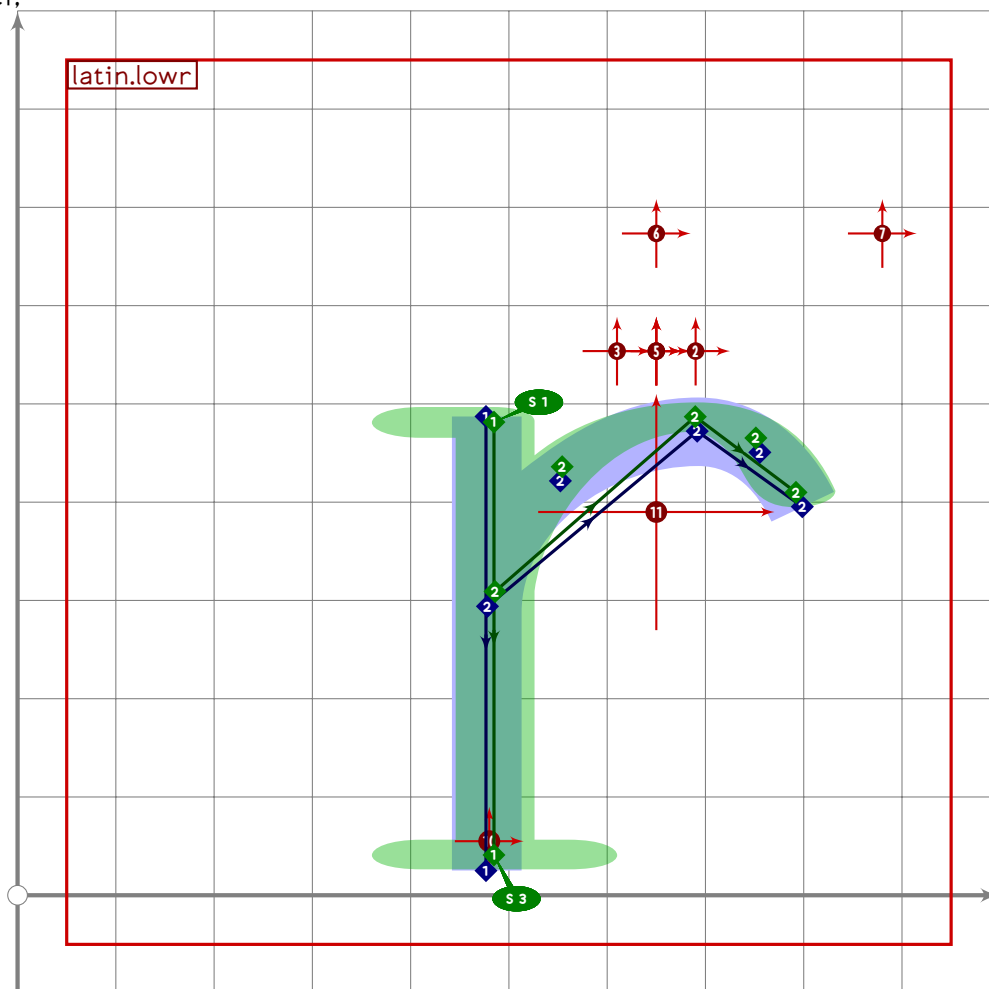
```


U+FF52
tsuku.uniFF52

```

1557 y2=y3=latin_wide_xheight_h;
1558 y4=0.47[y3,y5];
1559 y5=latin_wide_low_h;
1560 y6=0.91[y3,y5];
1561
1562 z0=z1+150*(dir 20);
1563
1564 push_stroke(z0-(0.6[z0,z1])-z1-z2{left}..
1565   {left}z3.{down}z4.{right}z5..z6,
1566   (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-
1567   (1.6,1.6)-(1.6,1.6)-(1.6,1.6));
1568 replace_strokep(0)(subpath (0,6.97) of oldp);
1569 set_botip(0,2,0);
1570 set_botip(0,3,1);
1571 if not do_italic_hook: set_boserif(0,3,2); fi;
1572
1573 tsu_accent.shift_anchors(ypart olda<vmetric(0.52))((-30,0));
1574 tsu_accent.shift_anchors(ypart olda>vmetric(0.52))((50,0));
1575 expand_pbox;
1576 enddef;

```



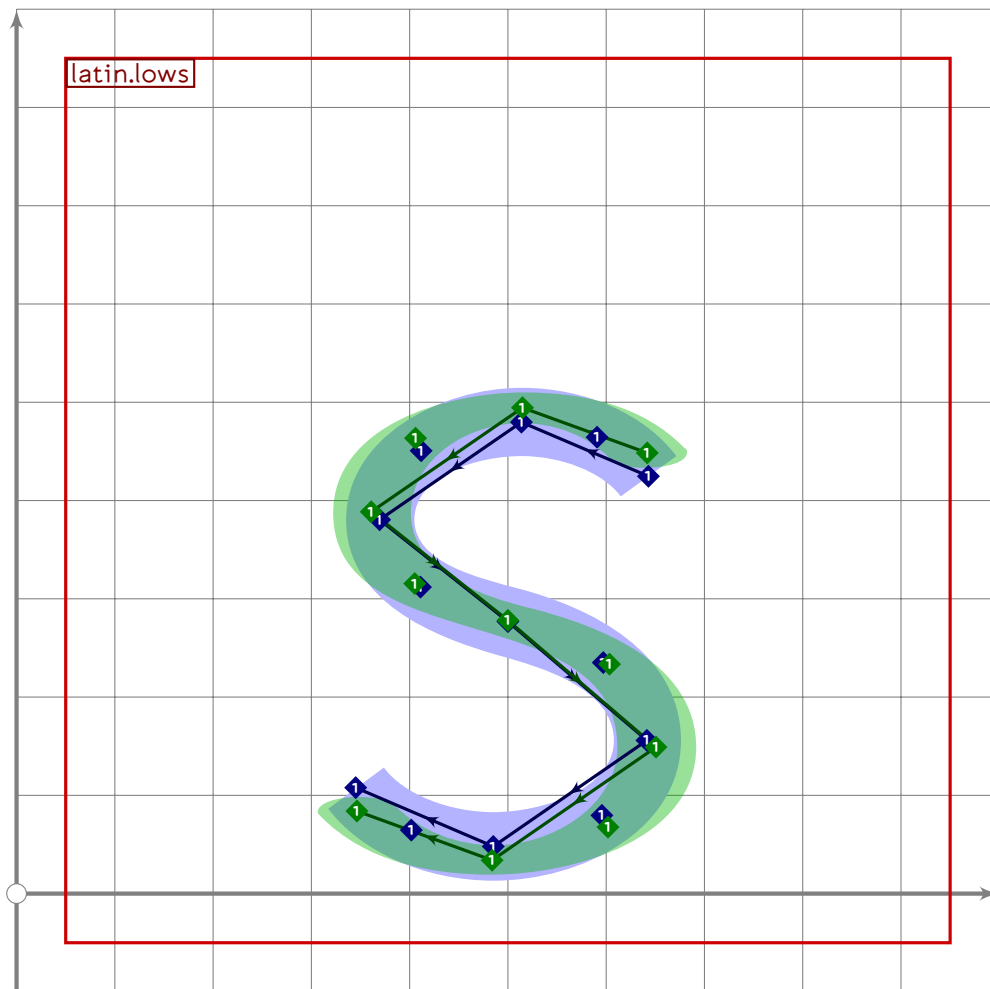
LATI

1577

```

1578 vardef latin.lowr =
1579   push_pbox_toexpand("latin.lowr");
1580   (x1+x5)/2=650;
1581   (x5-x1)=(y1-y2)*0.75;
1582   x2=x1=x3;
1583   x4=0.62[x3,x5];
1584
1585   y1=latin_wide_xheight_v;
1586   y2=latin_wide_low_v;
1587   y3=0.58[y2,y4];
1588   y4=0.5[latin_wide_xheight_h,latin_wide_xheight_r];
1589   y5=0.60[y2,y4];
1590
1591   push_stroke(z1-z2,(1.6,1.6)-(1.6,1.6));
1592   set_boserif(0,0;if do_italic_hook: 11 else: 1 fi);
1593   if not do_italic_hook: set_boserif(0,1,3); fi;
1594
1595   push_stroke(z3..z4{right}..{dir 273}z5,
1596     (1.6,1.6)-(1.6,1.6)-(1.6,1.6));
1597   replace_strokep(0)(subpath (0.03,1.6) of oldp);
1598
1599   tsu_accent.shift_anchors(ypart olda>vmetric(0.05))((0.5[x3,x5]-500,0));
1600   tsu_accent.shift_anchors(ai=anc_lower_connect)((-20,0));
1601   expand_pbox;
1602 enddef;

```



```

1603
1604 vardef latin.lows =
1605   push_pbox_toexpand("latin.lows");
1606   transform tatb;
1607   path mycurve;
1608
1609   mycurve:=(1,0)..(0,1)..(-1,0);
1610
1611   y2=latin_wide_xheight_r;
1612   y0=y3=0.77[y6,y2];
1613   y4=0.53[y6,y2];
1614   y5=y8=0.25[y6,y2];
1615   y6=latin_wide_low_r;
1616
1617   0.48[x1,x7]=0.48[x2,x6]=0.48[x3,x5]=x4=500;
1618   x5-x1=5;
1619   x5-x7=(y2-y6)*0.67;
1620
1621   (point 0 of mycurve) transformed ta=z0;
1622   (point 0.35 of mycurve) transformed ta=z1;
1623   (point 1 of mycurve) transformed ta=z2;

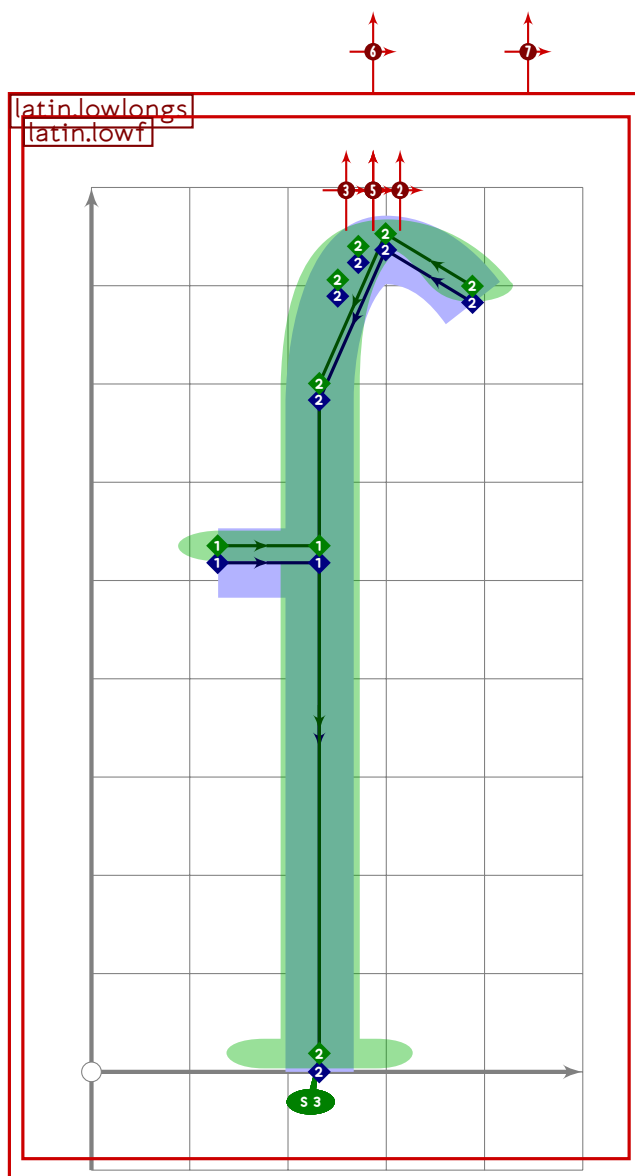
```

```

1624 (point 2 of mycurve) transformed ta=z3;
1625 xypart ta=0;
1626
1627 (point 0 of mycurve) transformed tb=z8;
1628 (point 0.35 of mycurve) transformed tb=z7;
1629 (point 1 of mycurve) transformed tb=z6;
1630 (point 2 of mycurve) transformed tb=z5;
1631
1632 if sharp_corners:
1633   mycurve:=subpath (0.29,2) of mycurve;
1634 else:
1635   mycurve:=subpath (0.38,2) of mycurve;
1636 fi;
1637
1638 push_stroke((mycurve transformed ta)..z4..(reverse mycurve transformed tb),
1639   (1.6,1.6)–(1.6,1.6)–(1.6,1.6)–(1.6,1.6)–(1.6,1.6)–
1640   (1.6,1.6)–(1.6,1.6));
1641 expand_pbox;
1642 endif;

```

U+017F
tsuku.long

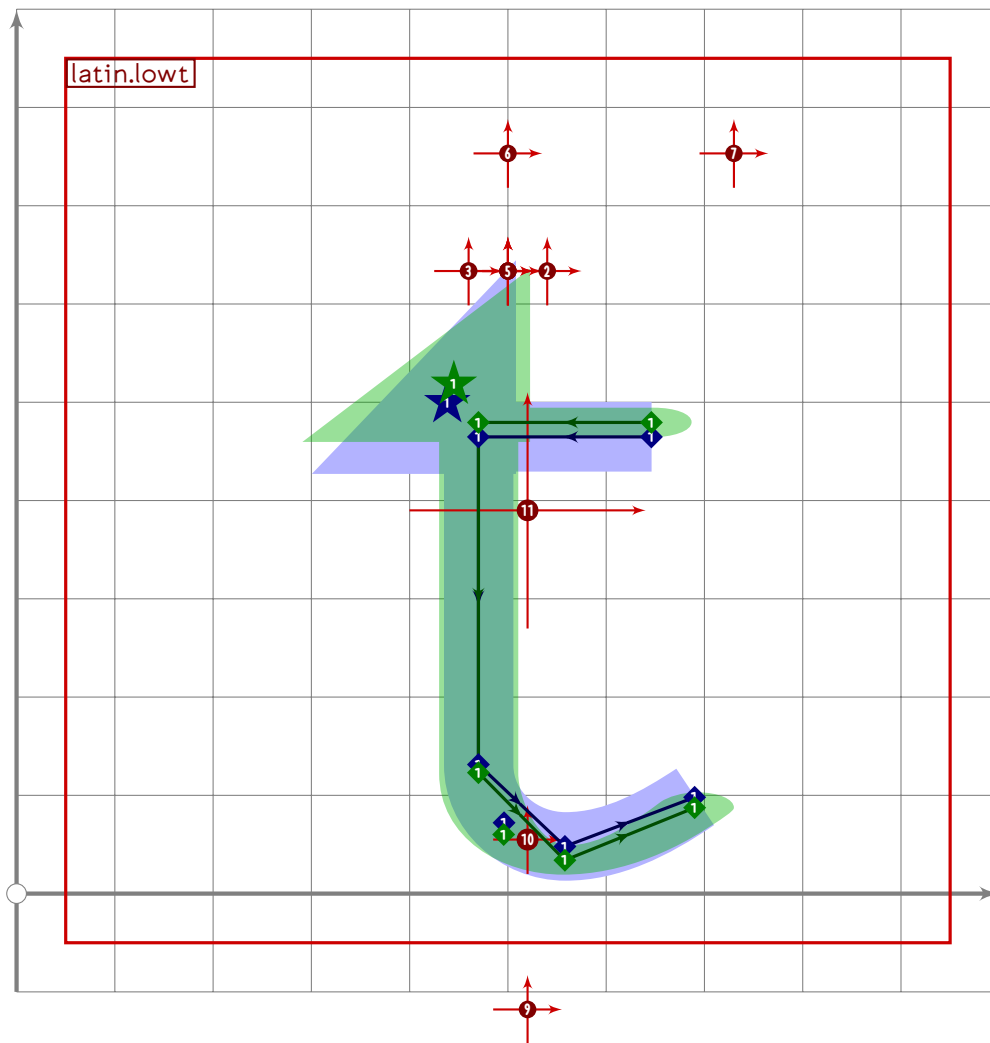


```

1643
1644 vardef latin.lowlongs =
1645   push_pbox_toexpand("latin.lowlongs");
1646   latin.lowf;
1647   replace_strokep(-1)(subpath (0,0.52) of oldp);
1648   expand_pbox;
1649 enddef;

```

LATI



```

1650
1651 vardef latin.lowt =
1652   push_pbox_toexpand("latin.lowt");
1653   x2=x3=x10=470;
1654   x1=0.2[x5,x2];
1655   x4=0.6[x5,x2];
1656   x5-x2=220;
1657
1658   y1=y2=y6=latin_wide_xheight_h;
1659   y3=0.2[y4,y1];
1660   y4=latin_wide_low_r;
1661   y5=0.12[y4,y1];
1662   y10=vmetric(0.83);
1663
1664   if is_proportional:
1665     z10-z6=whatever*dir 58;
1666   else:
1667     z10-z6=whatever*dir 47;
1668   fi;
1669

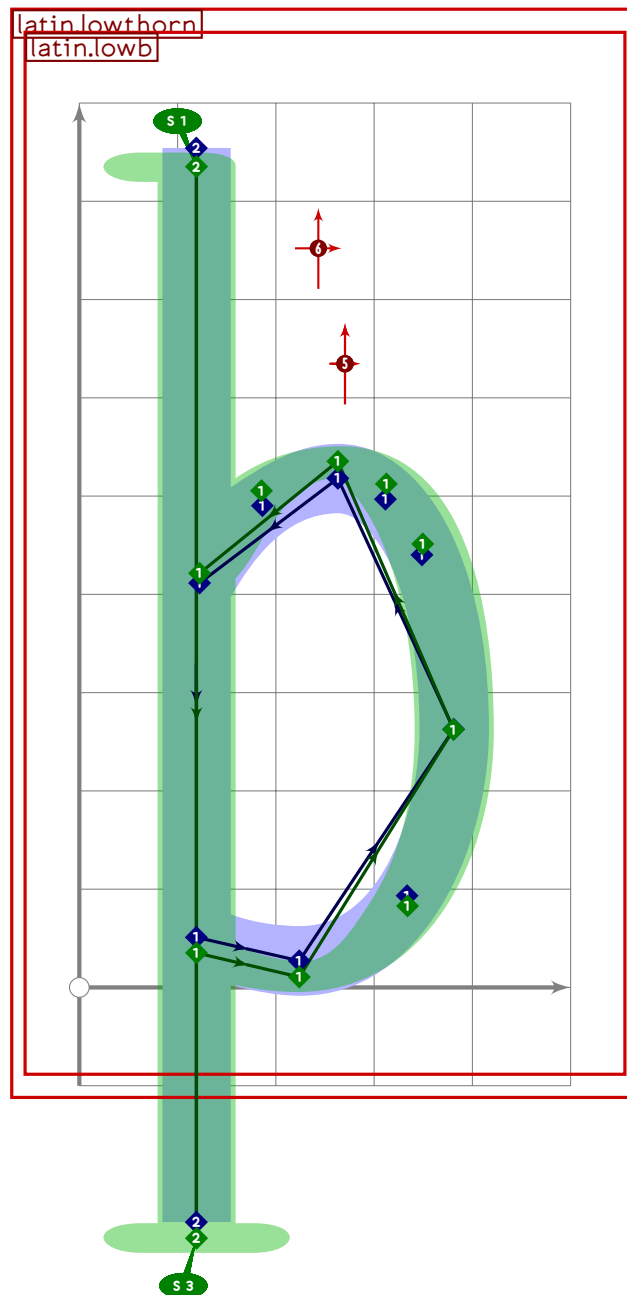
```

LATI

```

1670 if tsu_brush_max.brpunct*100<30:
1671     push_stroke(z1-z6-z10-z3{down}..z4{right}..{curl 0.2}z5,
1672         (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
1673     set_botip(0,1,1);
1674     set_botip(0,2,1);
1675 else:
1676     push_stroke(z1-z2-z3{down}..z4{right}..{curl 0.2}z5,
1677         (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
1678     set_botip(0,1,0);
1679 fi;
1680
1681 push_stroke(z6-z1,(0,0)-(0,0));
1682
1683 x7=x8=x2;
1684 x9=x6;
1685
1686 y7=y9=y2;
1687
1688 if is__proportional:
1689     z8-z9=whatever*dir 58;
1690 else:
1691     z8-z9=whatever*dir 47;
1692 fi;
1693
1694 if tsu_brush_max.brpunct>=0.3:
1695     begingroup
1696         save t; transform t;
1697         t:=tsu_rescale_xform;
1698         push_lcblob(((z7 transformed t)+(mbrush_width,-mbrush_height))-
1699             ((z8 transformed t)+(mbrush_width,mbrush_height))-
1700             ((z9 transformed t)+(-mbrush_width,-mbrush_height))-cycle);
1701         replace_lcblob(0)(oldblob transformed inverse t);
1702     endgroup;
1703 fi;
1704
1705 tsu_accent.shift_anchors(ypart olda>vmetric(0.52))
1706     ((0,vmetric(0.12)-vmetric(0)));
1707 tsu_accent.shift_anchors(ypart olda<vmetric(0.52))
1708     ((20,0));
1709 expand_pbox;
1710 endif;

```



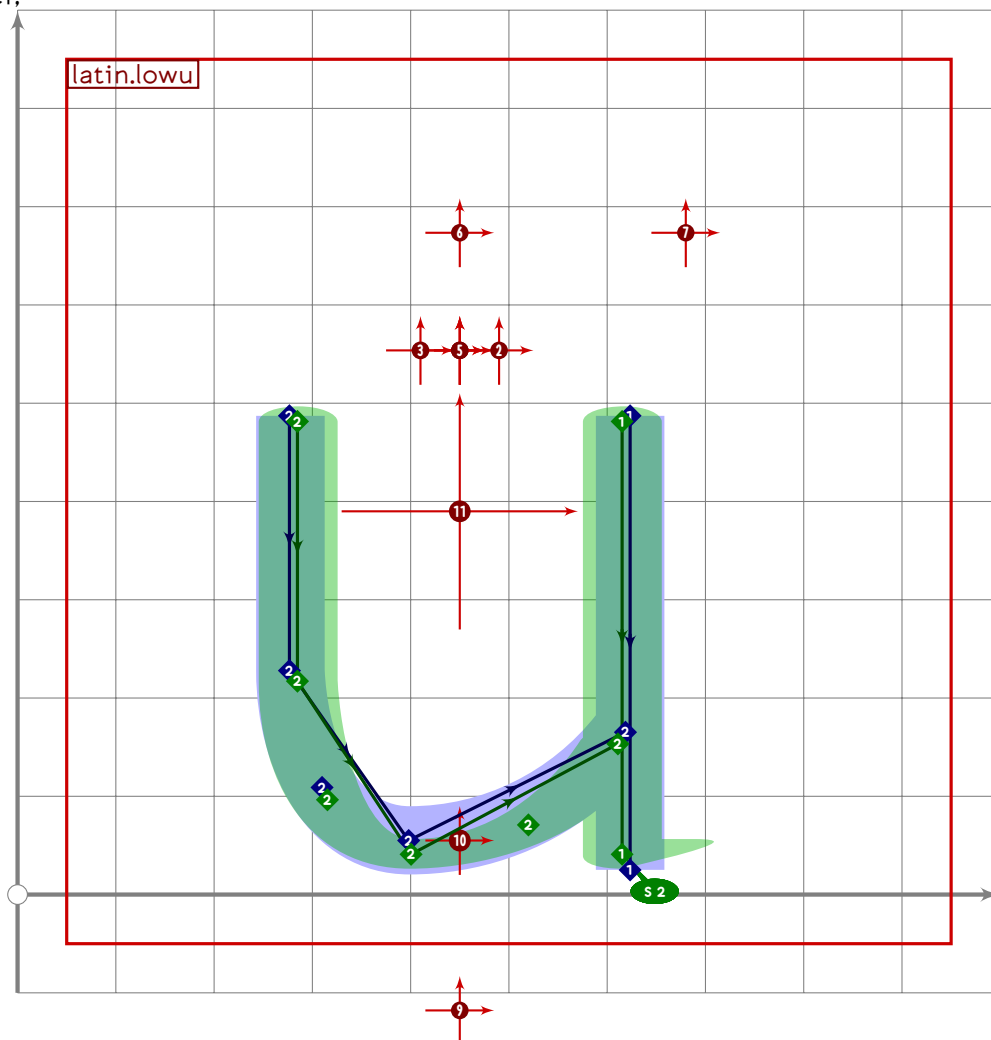
```

1711
1712 vardef latin.lowthorn =
1713   push_pbox_toexpand("latin.lowthorn");
1714   latin.lowb;
1715   set_botip(0,whatever);
1716   set_boserif(0,0,whatever);
1717   push_stroke((point 0 of get_strokep(0))--
1718     (xpart point 0 of get_strokep(0),latin_wide_desc_v,
1719     (1.6,1.6)--(1.6,1.6)));
1720   set_boserif(0,0,1);
1721   set_boserif(0,1,3);
1722   replace_strokep(-1)(subpath (1,infinity) of oldp);
1723   replace_strokeq(-1)(subpath (1,infinity) of oldq);
1724   expand_pbox;

```


U+FF55
tsuku.uniFF55

1725 enddef;



```

1726
1727 vardef latin.lowu =
1728   push_pbox_toexpand("latin.lowu");
1729   (x1+x5)/2=450;
1730   (x1-x5)=(y2-y1)*0.75;
1731   x2=x1=x3;
1732   x4=0.65[x3,x5];
1733   x6=x5;
1734
1735   y1=latin_wide_low_v;
1736   y2=y6=latin_wide_xheight_v;
1737   y3=0.73[y2,y4];
1738   y4=latin_wide_low_h;
1739   y5=0.60[y2,y4];
1740
1741   push_stroke(z2-z1,(1.6,1.6)-(1.6,1.6));
1742   set_boserif(0,if do_italic_hook: 11 else: 2 fi);
1743
1744   push_stroke(reverse(z3..z4{left}..z5{dir 93}-z6),

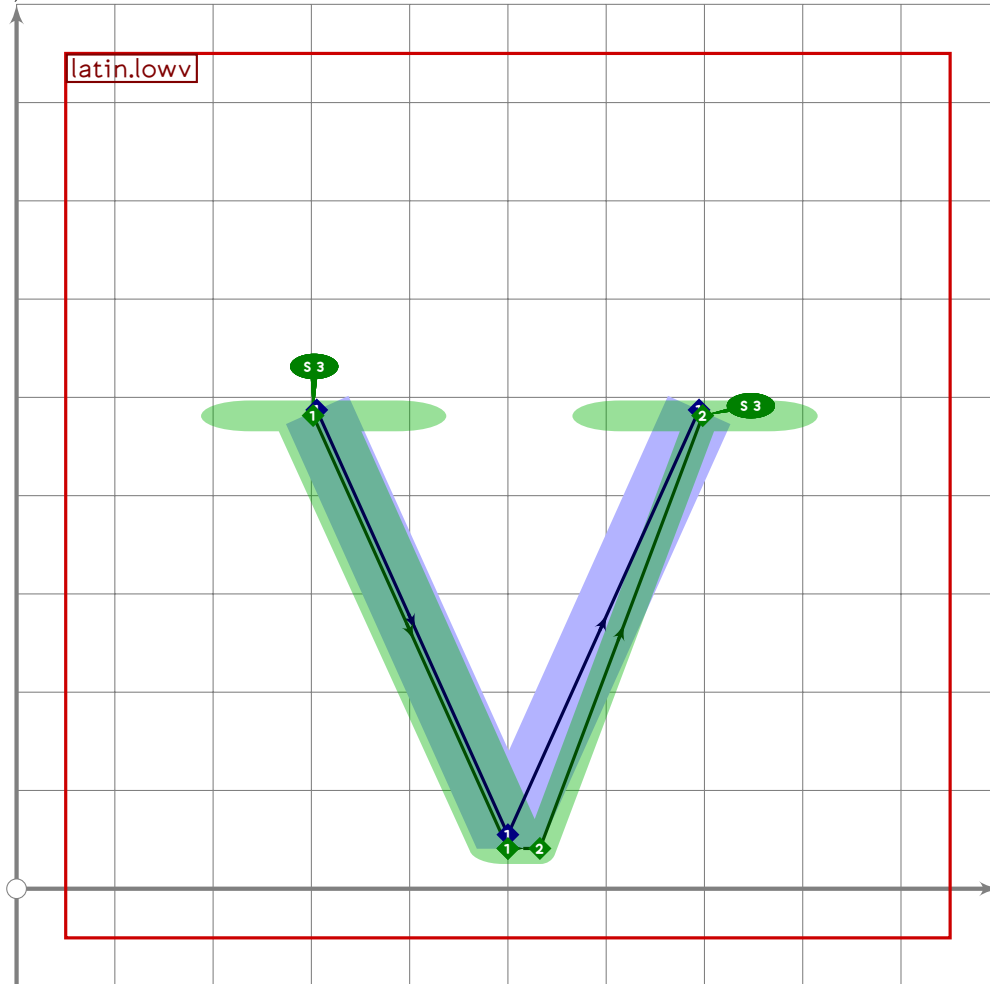
```

LATI

```

1745     (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
1746 replace_strokep(0)(subpath (0,2.97) of oldp);
1747 if do_italic_hook: set_boserif(0,0,11); fi;
1748
1749 tsu_accent.shift_anchors(true)((-50,0));
1750 expand_pbox;
1751 enddef;

```



```

1752
1753 vardef latin.lowv =
1754   push_pbox_toexpand("latin.lowv");
1755   (x1+x3)/2=x2=500;
1756
1757   y1=y3=latin_wide_xheight_v;
1758   y2=latin_wide_low_h;
1759
1760   (x3-x1)=(y1-y2)*0.9;
1761
1762   if do_alteration:
1763     push_stroke(z1-z2,(1.6,1.6)-(1.6,1.6));
1764     set_boserif(0,0,3);
1765

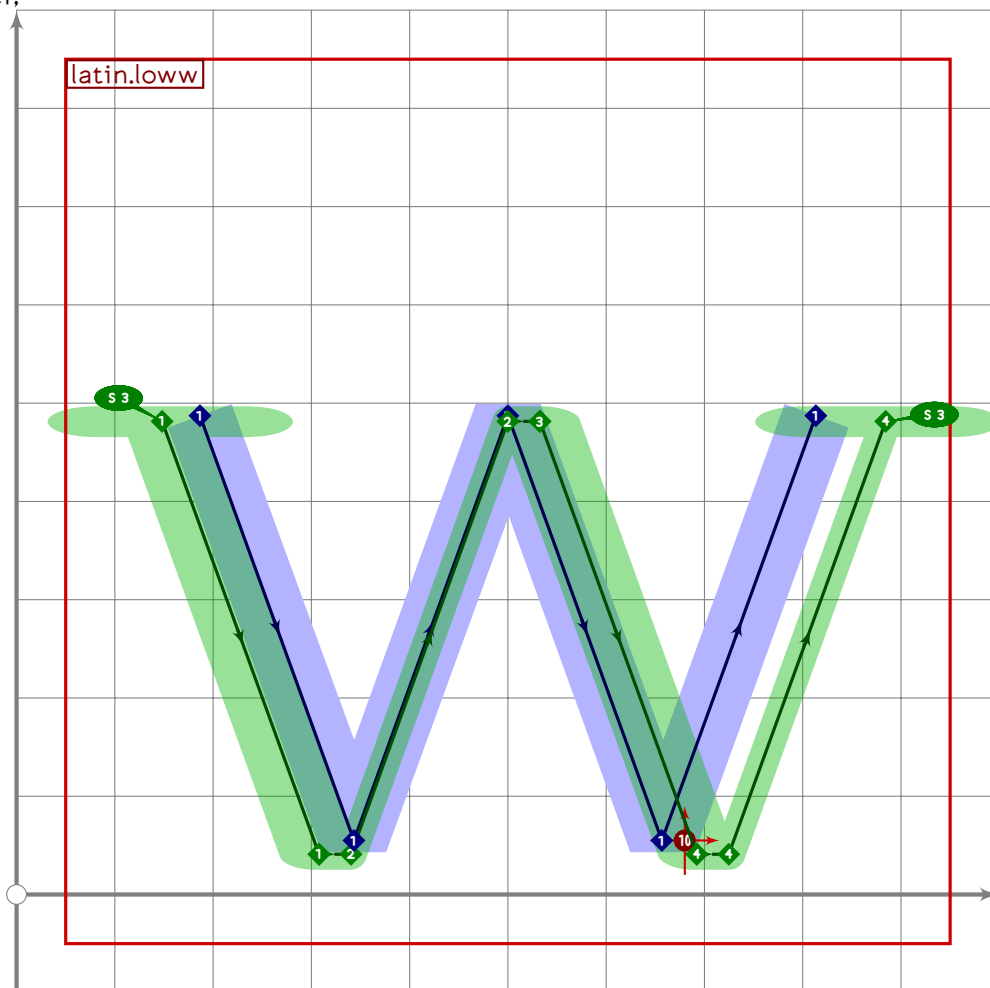
```

U+FF57
tsuku.uniFF57

```

1766   push_stroke(z2-(z2+alternate_adjust*right)-z3,
1767     (1.6,1.6)-(1.6,1.6)-(1.6,1.6));
1768   set_boserif(0,2,3);
1769   set_bobrush(0,bralternate);
1770   else:
1771     push_stroke(z1-z2-z3,(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
1772     set_botip(0,1,0);
1773     set_boserif(0,0,1);
1774   fi;
1775   expand_pbox;
1776 enddef;

```



```

1777
1778 vardef latin.loww =
1779   push_pbox_toexpand("latin.loww");
1780   (x1+x5)/2=(x2+x4)/2=x3=500;
1781   (x3-x2)=(x2-x1);
1782
1783   y1=y3=y5=latin_wide_xheight_v;
1784   y2=y4=latin_wide_low_h;
1785
1786   (x5-x1)=(y1-y2)*1.45;

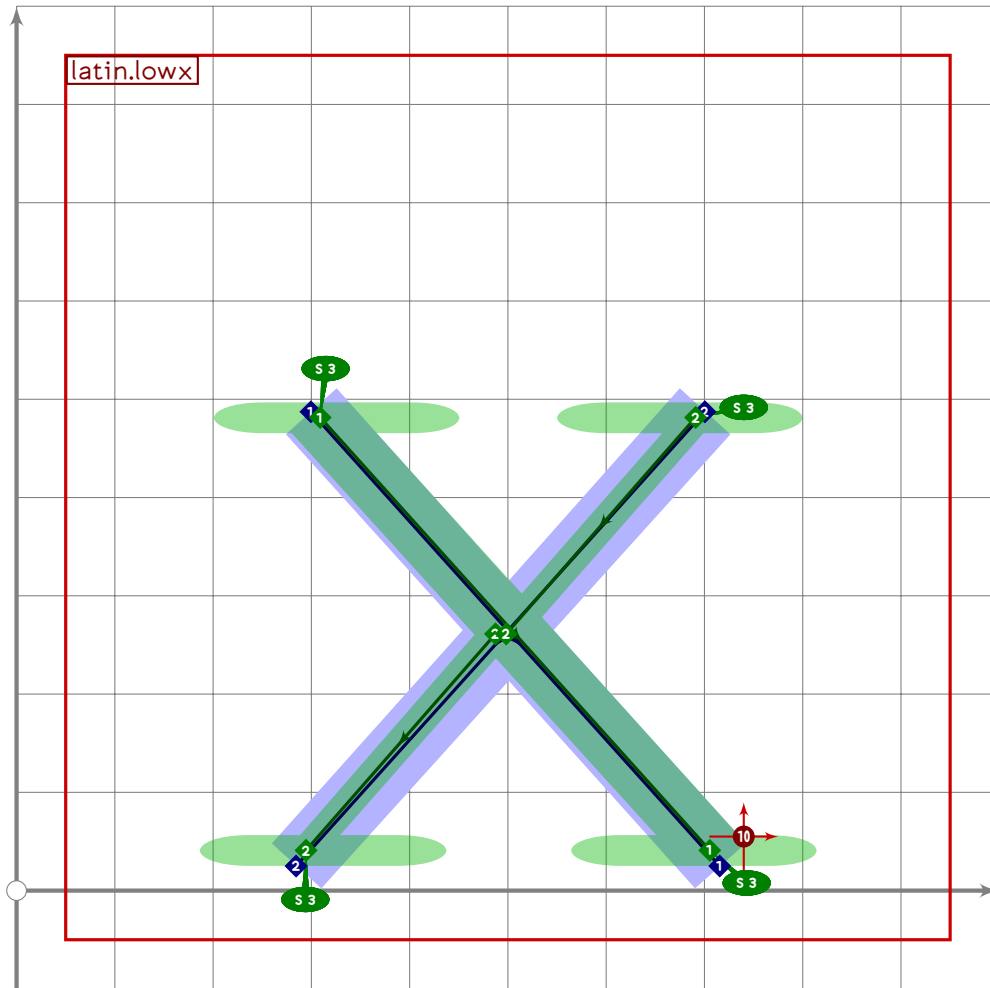
```

LATI

```

1787
1788 if do_alternation:
1789     push_stroke((z1-z2) shifted (alternate_adjust*left),
1790         (1.6,1.6)-(1.6,1.6));
1791     set_boserif(0,0,3);
1792
1793     push_stroke((z2+alternate_adjust*left)-z2-
1794         z3-(z3+alternate_adjust*right),
1795         (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
1796     set_bobrush(0,bralternate);
1797
1798     push_stroke((z3-z4) shifted (alternate_adjust*right),
1799         (1.6,1.6)-(1.6,1.6));
1800
1801     push_stroke((z4+alternate_adjust*right)-
1802         (z4-z5) shifted (alternate_adjust*right*2),
1803         (1.6,1.6)-(1.6,1.6)-(1.6,1.6));
1804     set_boserif(0,2,3);
1805     set_bobrush(0,bralternate);
1806 else:
1807     push_stroke(z1-z2-z3-z4-z5,
1808         (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
1809     set_botip(0,1,0);
1810     set_botip(0,2,0);
1811     set_botip(0,3,0);
1812     set_boserif(0,0,1);
1813 fi;
1814
1815 tsu_accent.shift_anchors(ai=anc_lower_connect)((180,0));
1816 expand_pbox;
1817 endif;

```



```

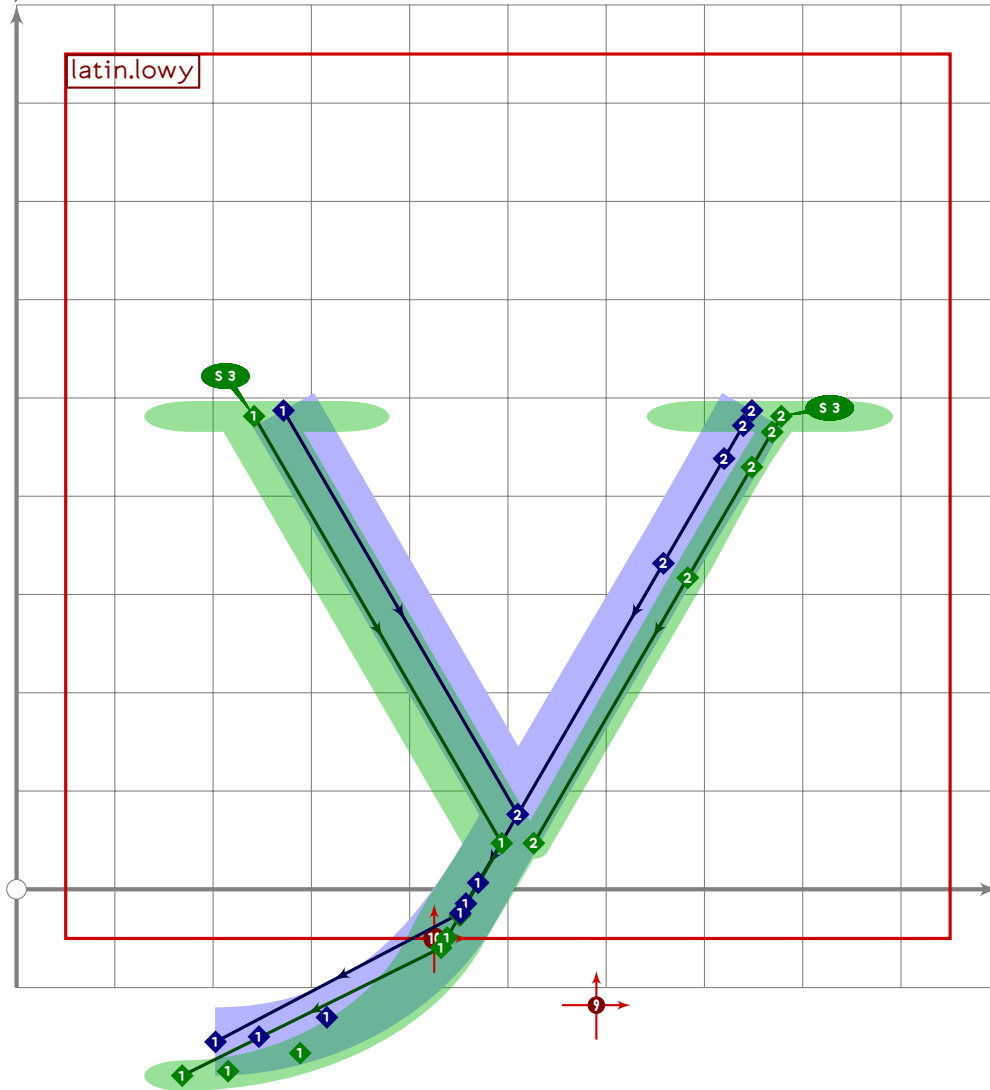
1818
1819 vardef latin.lowx =
1820   push_pbox_toexpand("latin.lowx");
1821   (x1+x3)/2=500;
1822   (x2+x4)/2=500;
1823   (x2+x3-x1-x4)=((y1-y2)*0.9)*2;
1824   (x3-x1)=(x2-x4)*0.93;
1825
1826   y1=y3=latin_wide_xheight_v;
1827   y2=y4=latin_wide_low_v;
1828
1829   push_stroke(z1-z2,(1.6,1.6)-(1.6,1.6));
1830   set_boserif(0,0,if do_italic_hook: 11 else: 3 fi);
1831   set_boserif(0,1,if do_italic_hook: 11 else: 3 fi);
1832
1833   if do_alteration:
1834     push_stroke(z3-(0.5[z3,z4]+alternate_adjust*right/6)
1835       -(0.5[z3,z4]+alternate_adjust*left/6)-z4,
1836       (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
1837     set_boserif(0,0,if do_italic_hook: 2 else: 3 fi);
1838     set_boserif(0,3,if do_italic_hook: 1 else: 3 fi);

```

```

1839 else:
1840   push_stroke(z3-z4,(1.6,1.6))-(1.6,1.6));
1841   set_boserif(0,0,if do_italic_hook: 2 else: 3 fi);
1842   set_boserif(0,1,if do_italic_hook: 1 else: 3 fi);
1843   fi;
1844   set_bobrush(0,bralternate);
1845
1846   tsu_accent.shift_anchors(ai=anc_lower_connect)((240,0));
1847   expand_pbox;
1848 enddef;

```



```

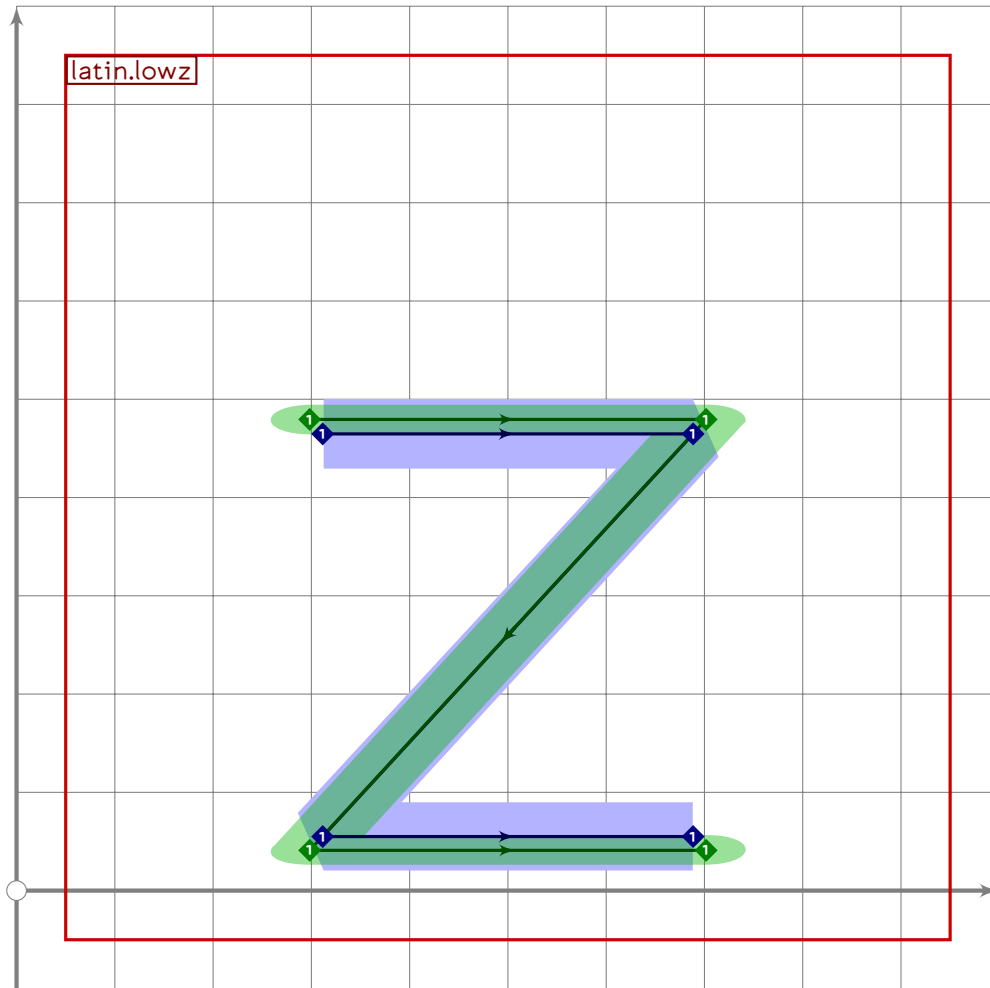
1849
1850 vardef latin.lowy =
1851   push_pbox_toexpand("latin.lowy");
1852   (x1+x3)/2=(x2+x4)/2=510;
1853   (x2+x3-x1-x4)=((y1-y2)*0.58)*2;
1854   (x3-x1)=(x2-x4)*0.93;
1855   x5=x4-0.1*(x2-x4);
1856

```

```

1857 y1=y3=latin_wide_xheight_v;
1858 y2=y4;
1859 y5=0.5[y4,latin_wide_low_h]=latin_wide_desc_h;
1860
1861 push_stroke(z1-z2,(1.6,1.6)-(1.6,1.6));
1862
1863 push_stroke(z3..tension 10..(0.6[z3,z4])..tension 0.8 and 3..{left}z5,
1864   (1.6,1.6)-(1.6,1.6)-(1.6,1.6));
1865
1866 numeric xchgtime;
1867 xchgtime:=ypart (get_strokep(-1) intersectiontimes get_strokep(0));
1868
1869 replace_strokep(-1)(z1-subpath (xchgtime,infinity) of get_strokep(0));
1870 replace_strokeq(-1)
1871   ((1.6,1.6)-subpath (xchgtime,infinity) of get_strokeq(0));
1872
1873 replace_strokep(0)(subpath (0,xchgtime) of oldp);
1874 replace_strokeq(0)(subpath (0,xchgtime) of oldq);
1875
1876 set_boserif(-1,0;if do_italic_hook: 11 else: 3 fi);
1877 set_boserif(0,0;if do_italic_hook: 2 else: 3 fi);
1878 set_botip(-1,1,1);
1879 set_bobrush(0,bralternate);
1880
1881 if do_alteration:
1882   replace_strokep(-1)(oldp shifted (alternate_adjust*left/2));
1883   replace_strokep(0)(oldp shifted (alternate_adjust*right/2));
1884 fi;
1885
1886 tsu_accent.shift_anchors(ai=anc_lower)((90,0));
1887 tsu_accent.shift_anchors(ai=anc_lower_connect)((-75,105));
1888 expand_pbox;
1889 enddef;

```



```

1890
1891 vardef latin.lowz =
1892   push_pbox_toexpand("latin.lowz");
1893   y1=y2=latin_wide_xheight_h;
1894   y3=y4=latin_wide_low_h;
1895
1896   x1=x3;
1897   x2=x4;
1898   (x1+x2)/2=500;
1899   (x2-x1)=(y1-y3)*0.92;
1900
1901   push_stroke(z1-z2-z3-z4,(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
1902   set_botip(0,1,0);
1903   set_botip(0,2,0);
1904   expand_pbox;
1905 enddef;

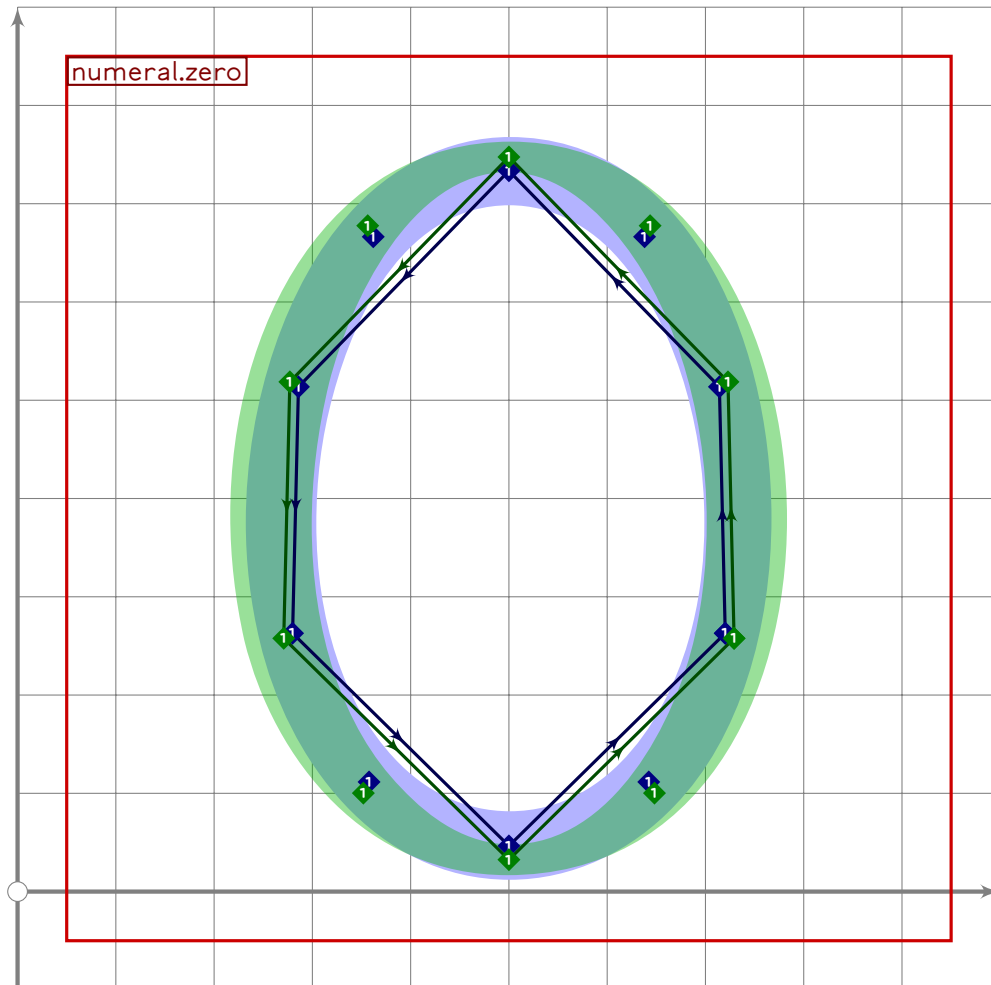
```


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(numerals);
32
33

```

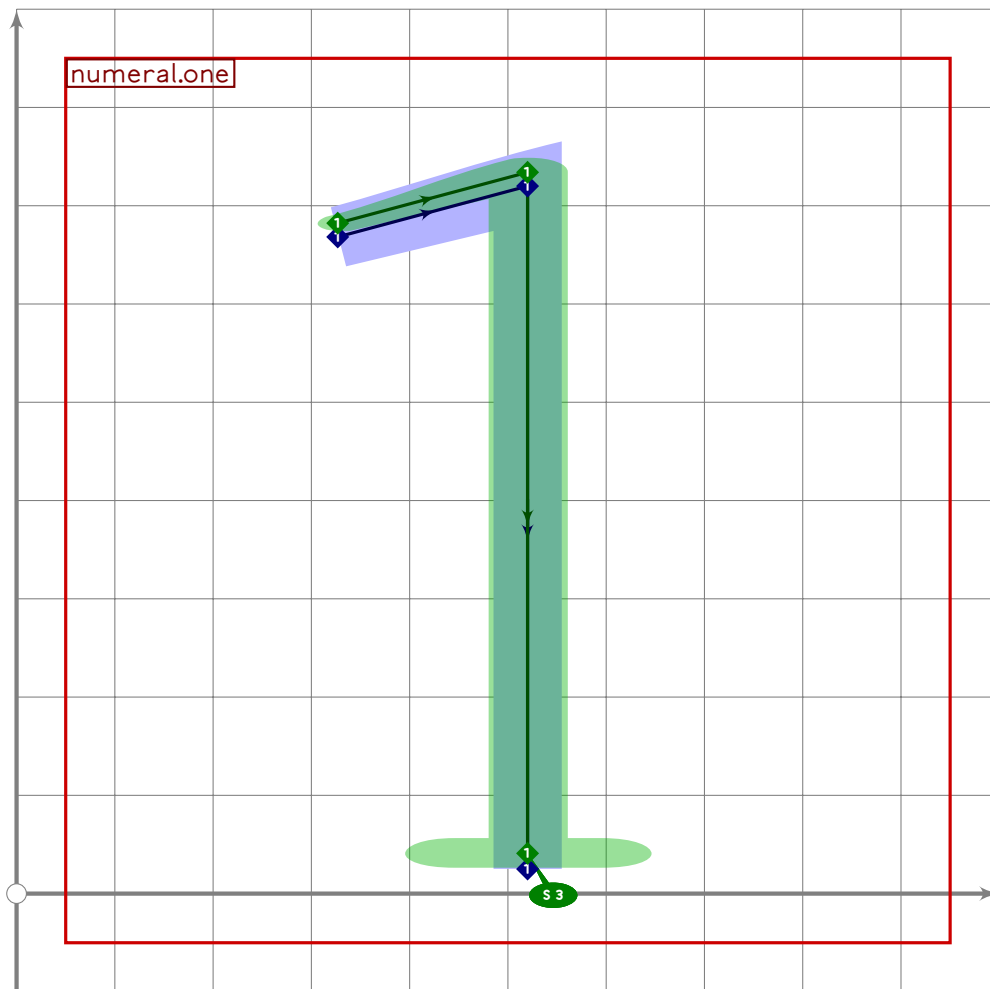


```

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

```

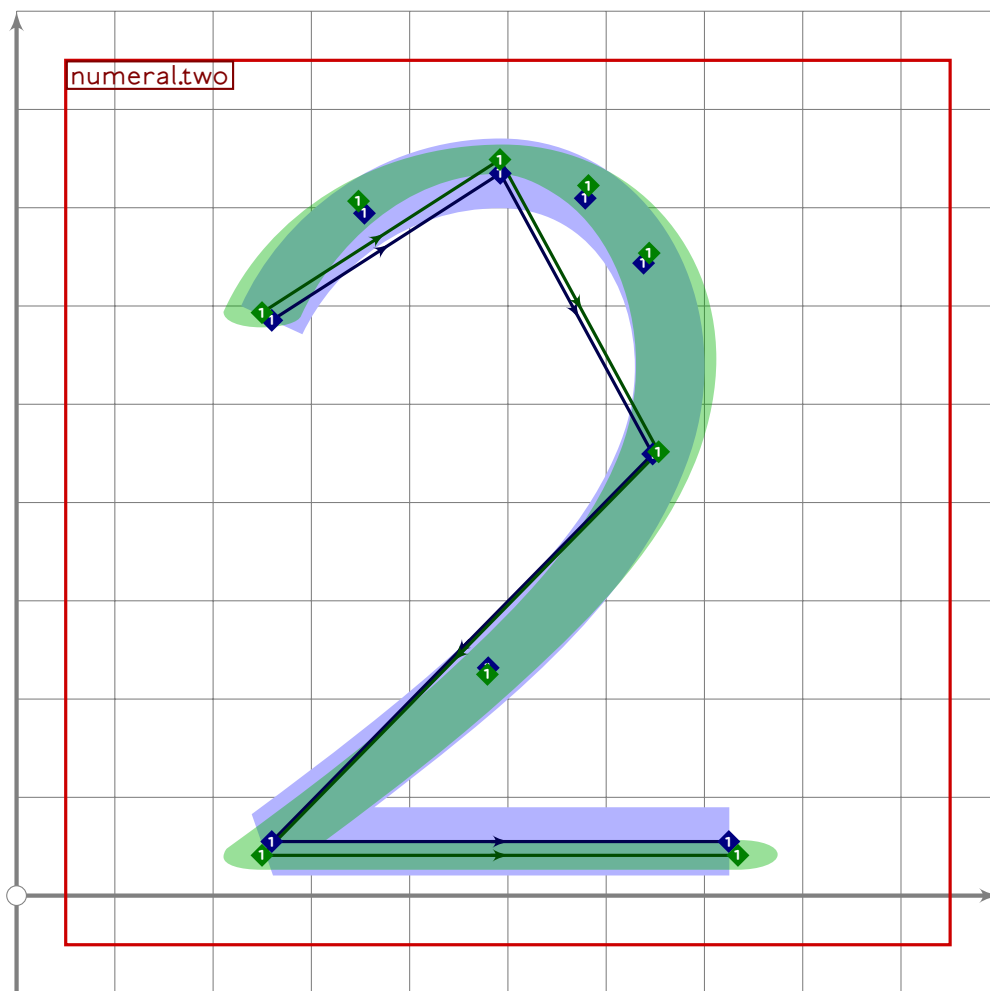
NUME



```

44
45 vardef numeral.one =
46   push_pbox_toexpand("numeral.one");
47   x3=x2=520;
48
49   y2=latin_wide_high_h;
50   y3=latin_wide_low_v;
51
52   z1=z2+200*dir 195;
53
54   push_stroke(z1-z2-z3,(1,1,1)-(1.6,1.6)-(1.6,1.6));
55   set_botip(0,1,1);
56   set_boserif(0,2,3);
57   expand_pbox;
58 enddef;

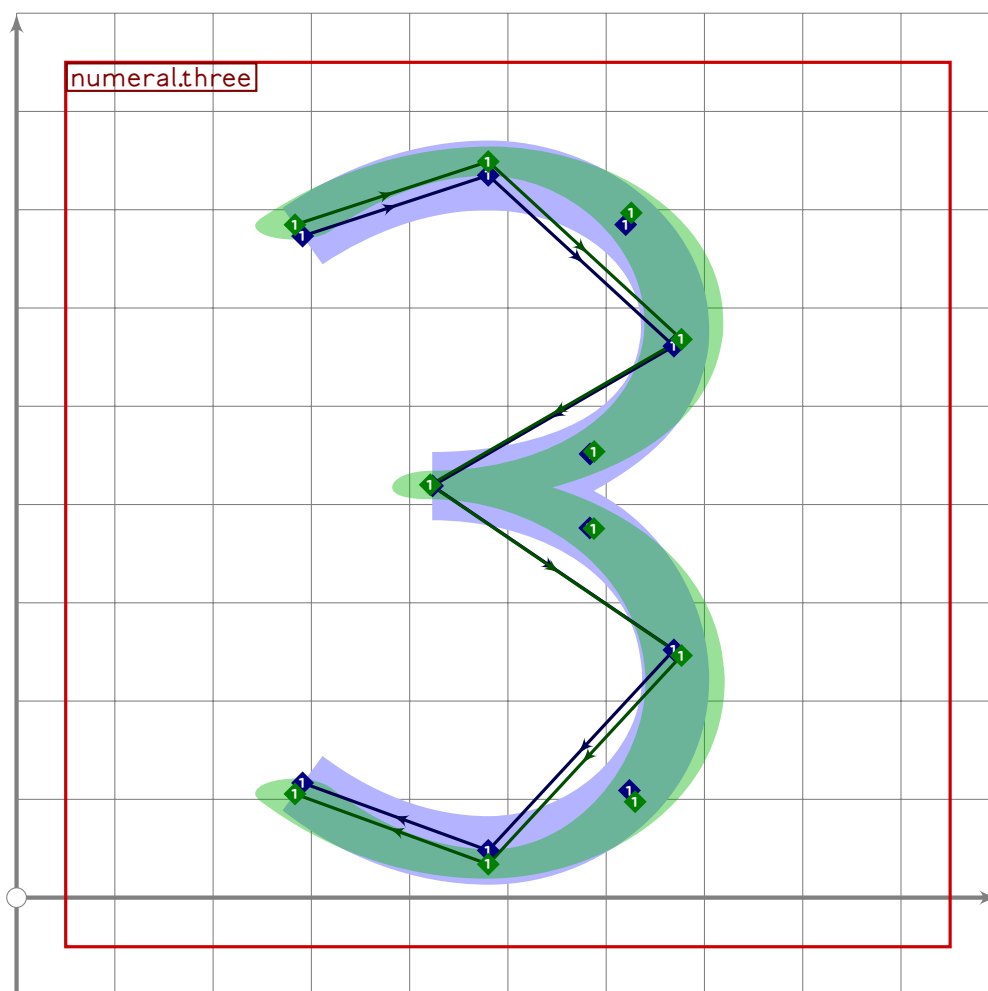
```



```

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

```



```

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

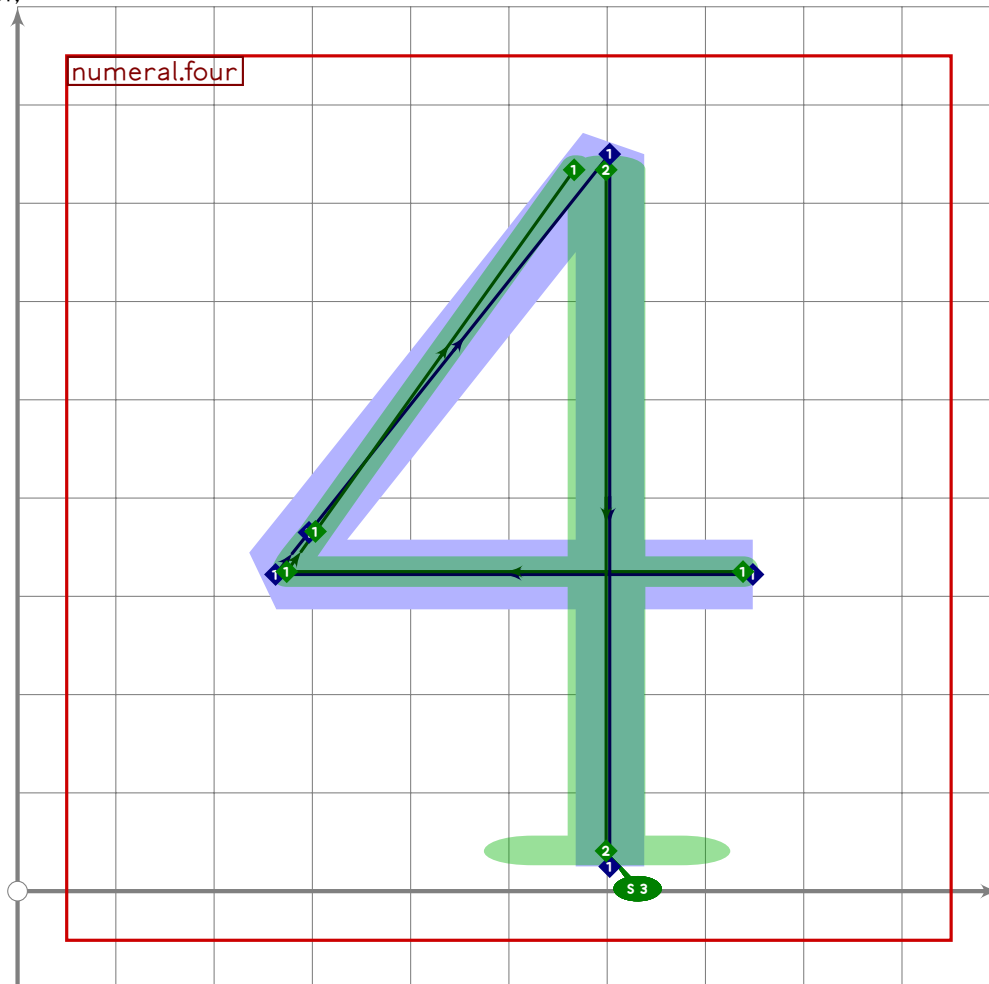
```

U+FF14
tsuku.uniFF14

```

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

```



```

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

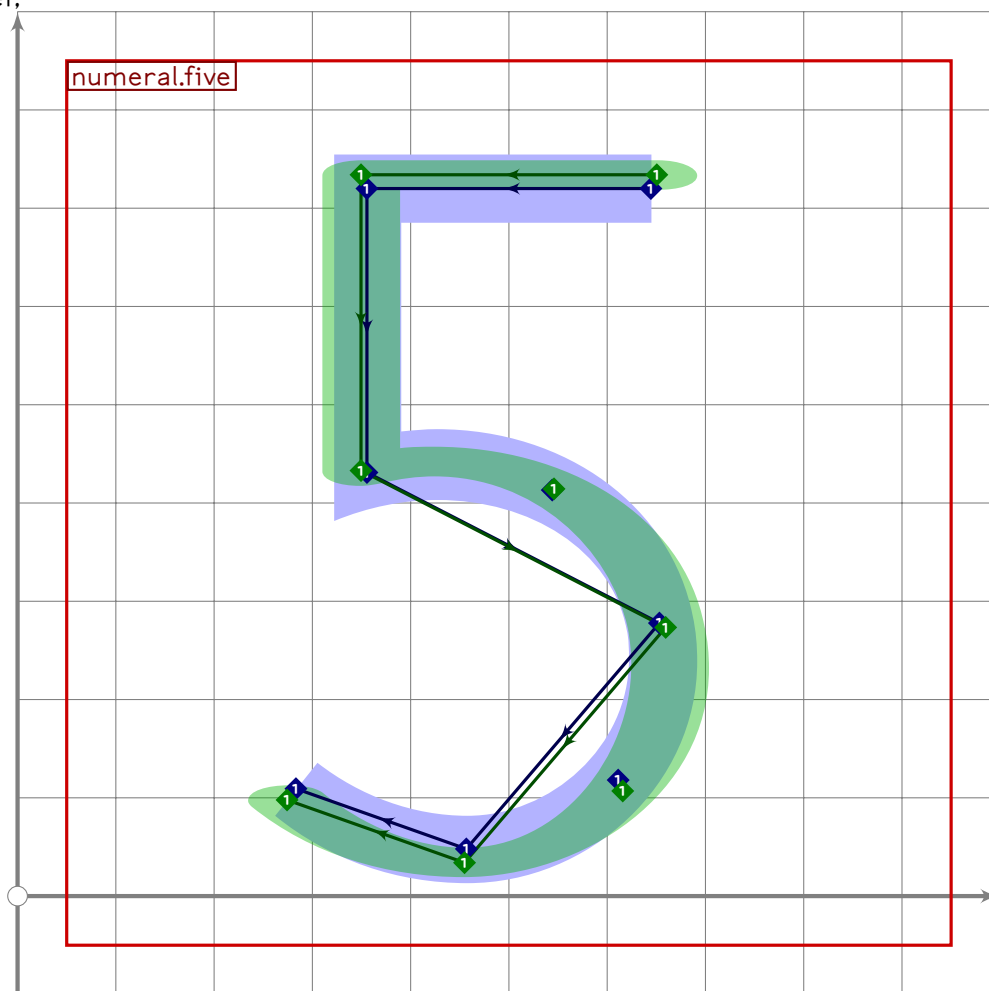
```

NUME

```

120   set_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;

```



```

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

```

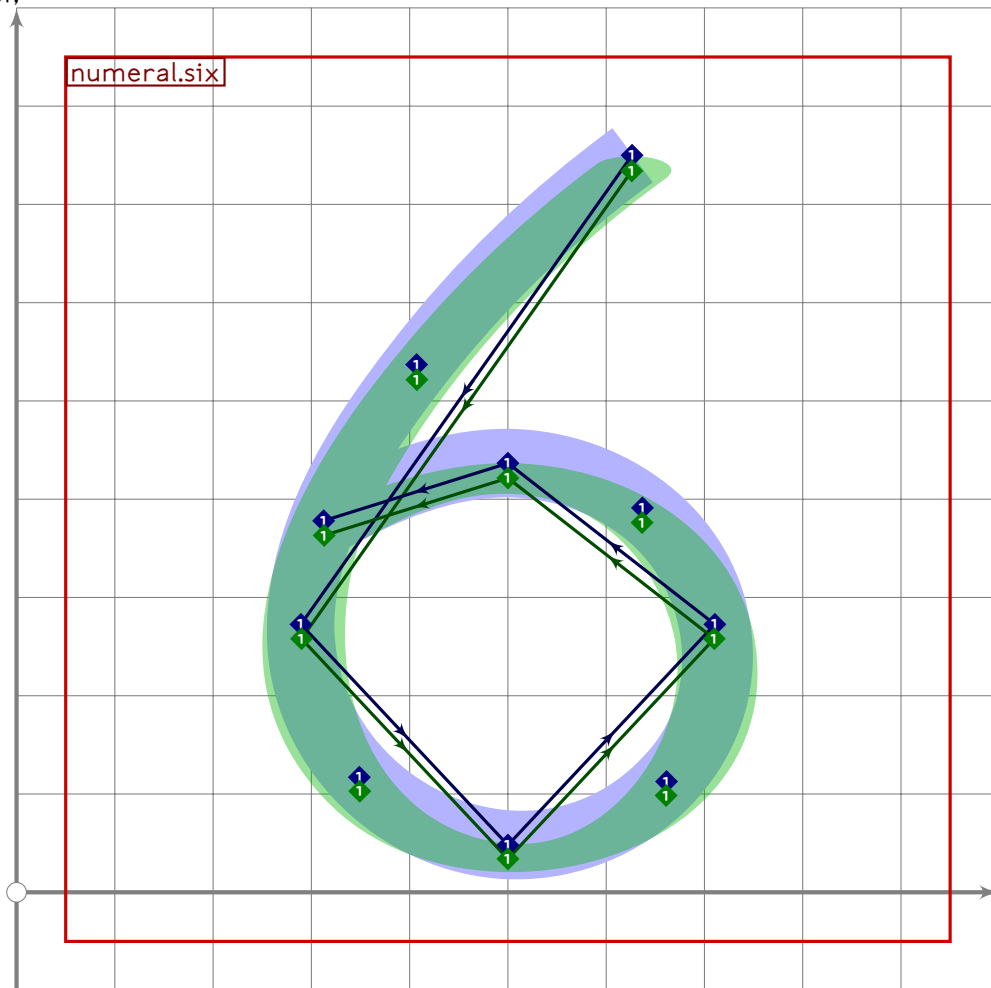
NUME

tsuku.uniFF16

```

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

```



NUME

```

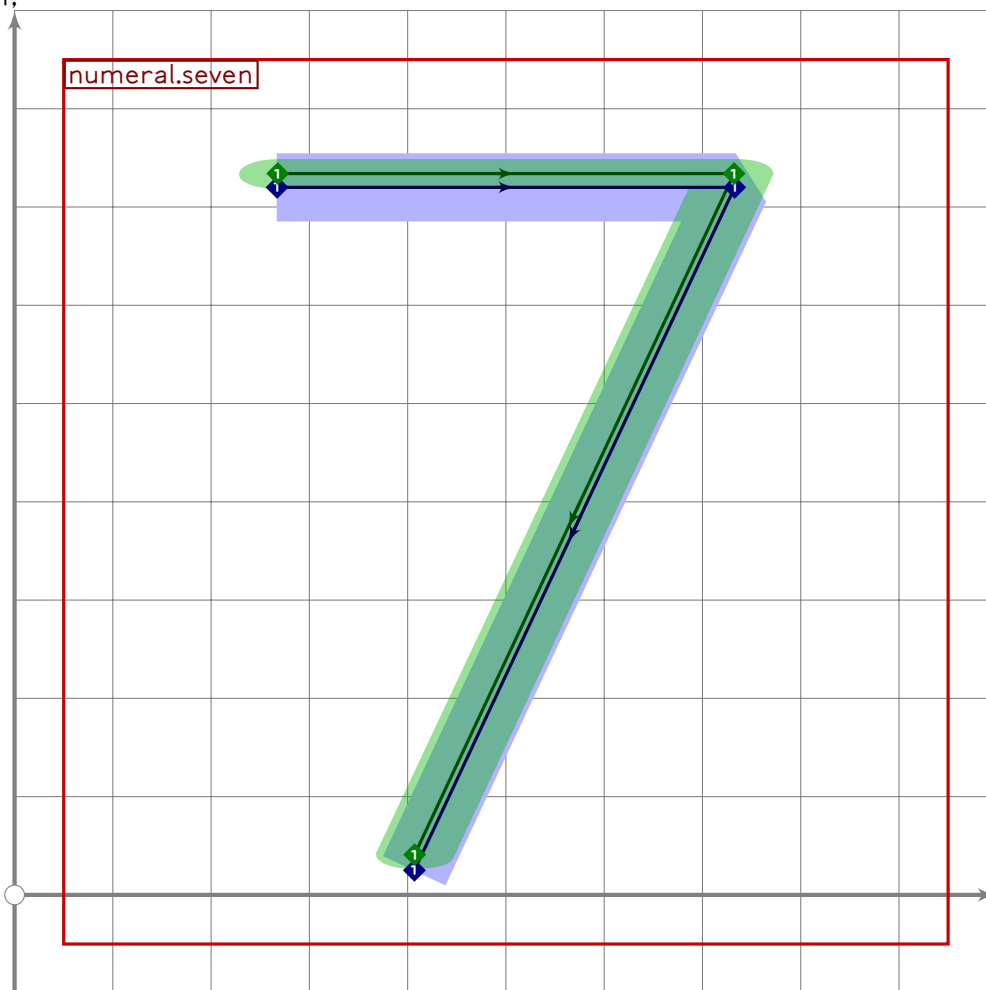
156
157 vardef numeral.six =
158   push_pbox_toexpand("numeral.six");
159   x1=0.8[x2,x4];
160   (x2+x4)/2=x3=500;
161   (x4-x2)=0.6(y1-y3);

```

```

162 x5=x3;
163
164 y1=latin_wide_high_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_stroke(0)(z1{curl 0.2}.tension 1.2..z2..z3..z4{dir 100}..
172   z5.{curl 0.2}(point 0.8 of oldp));
173 expand_pbox;
174 endif;

```



```

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

```

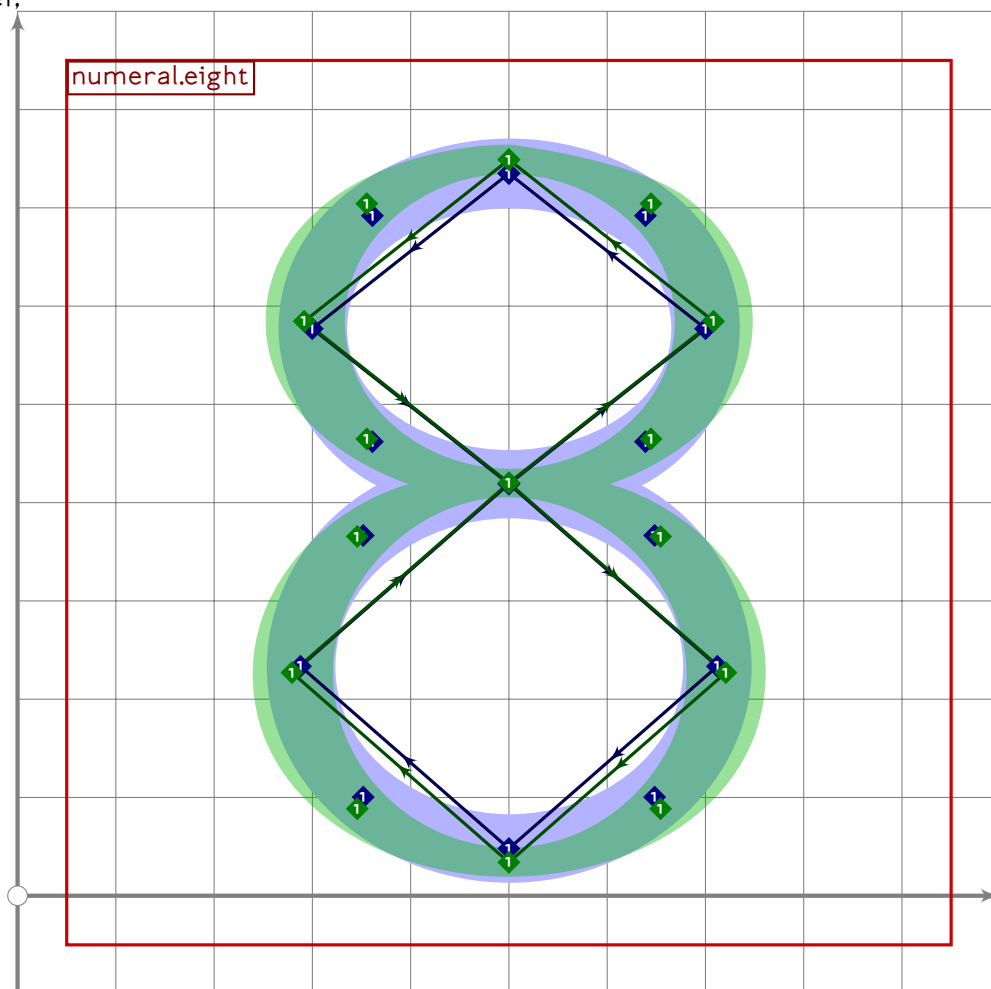
NUME

U+FF18
tsuku.uniFF18

```

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

```



```

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

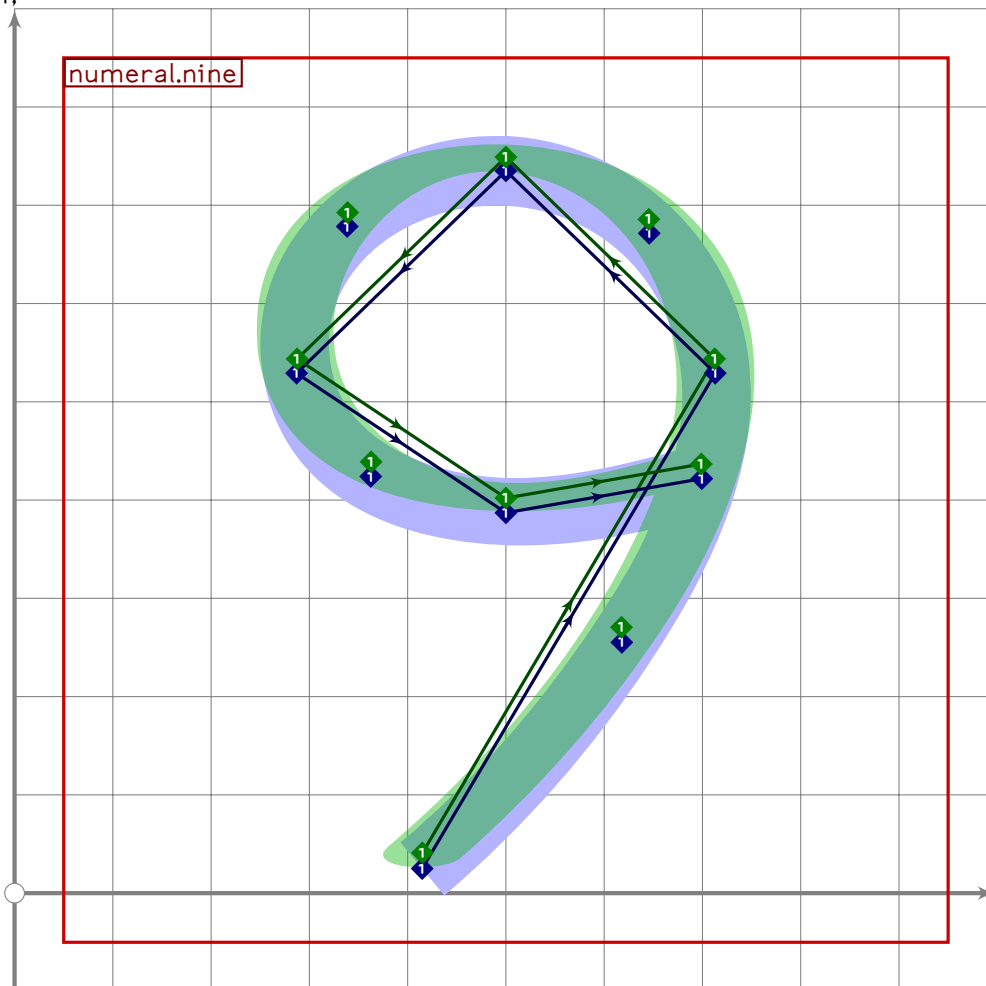
```

NUME

```

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

```



```

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

```

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

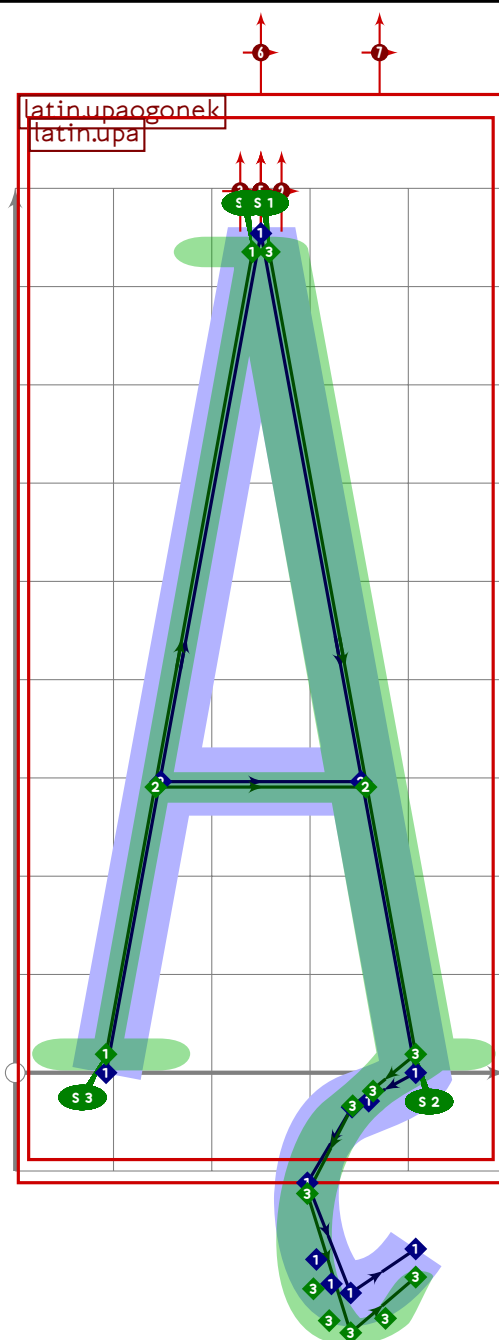
NUME

ogonek.mp

```

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

```

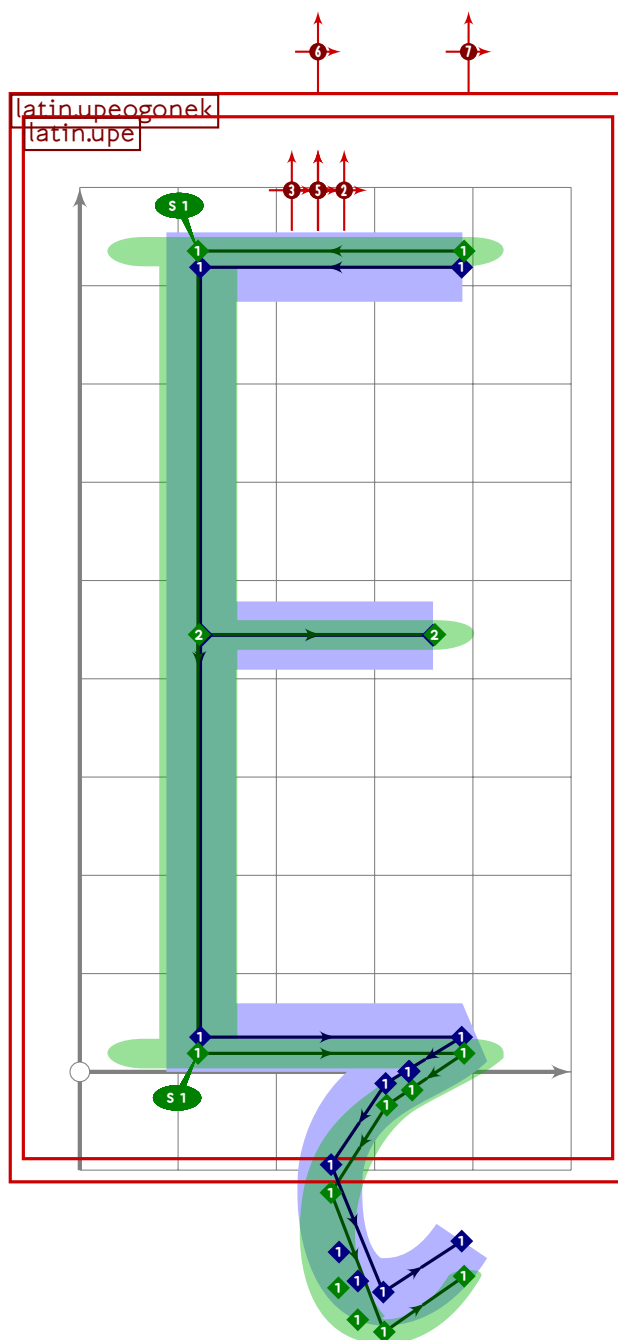


OGON

```

35 vardef latin.upaogonek =
36   push_pbox__toexpand("latin.upaogonek");
37   latin.upa;
38
39   x6=0.3[x2,x3];
40   x7=0.4[x6,x8];
41   x8=x3;
42
43   y6=0.5[y7,y3];
44   y7=latin_wide_desc_r;
45   y8=0.2[y7,y3];
46
47   if do_alteration:
48     replace_strokep(0)(oldp{dir 210}..z6..z7{right}..z8);
49     replace_strokep(0)(insert_nodes(oldp)(length(oldp)-2.5));
50     replace_strokeq(0)(oldq-(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];

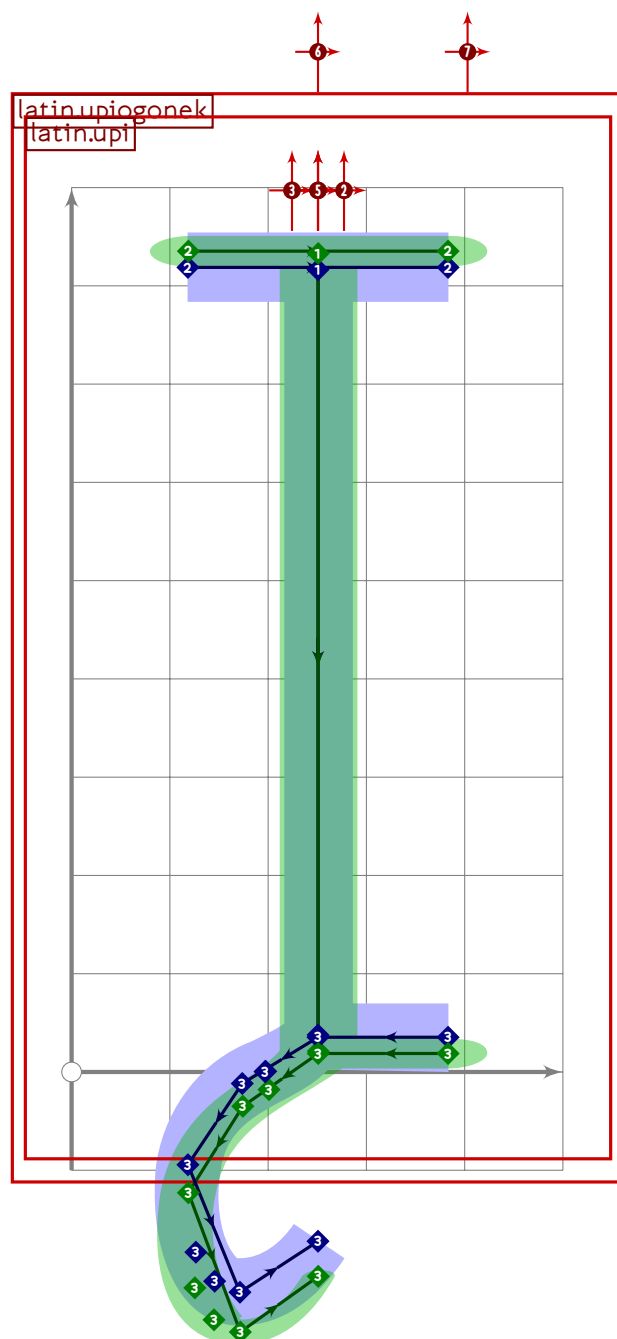
```

U+012E
tsuku.Iogonek

```

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-(1,4,1,4)-(1,3,1,3)-(1,4,1,4)-(1,1));
78  set_botip(-1,length(get_strokep(-1))-4,0);
79  expand_pbox;
80 enddef;

```



OGON

```

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

```

```

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

```


The diagram illustrates the stroke order and direction for writing the character 'Y'. The character is composed of two main parts: a vertical stem and a diagonal stroke. The vertical stem is formed by two parallel strokes, each labeled with a green diamond '1' and a blue diamond '1'. The diagonal stroke is formed by two parallel strokes, each labeled with a green diamond '2' and a blue diamond '2'. The diagram includes a grid, a red bounding box, and a legend for stroke order.

Legend:

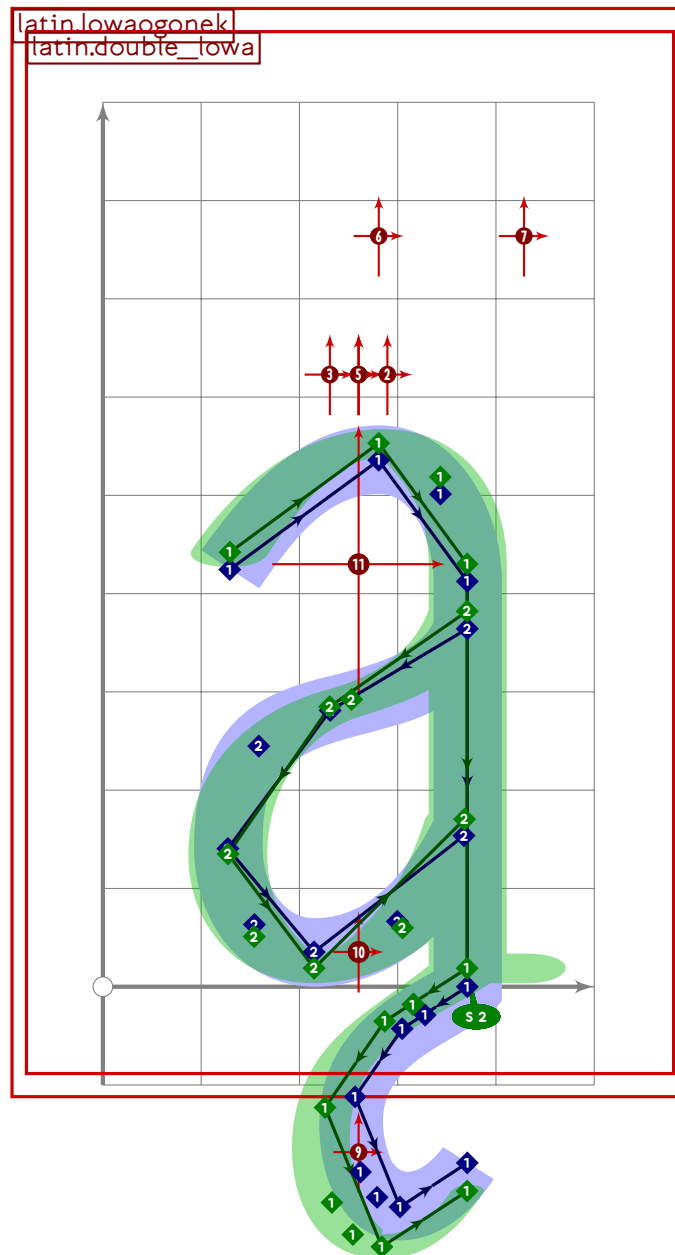
- Green diamond: Stroke order (1 or 2)
- Blue diamond: Stroke direction (1 or 2)

OGON

```

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 enddef;

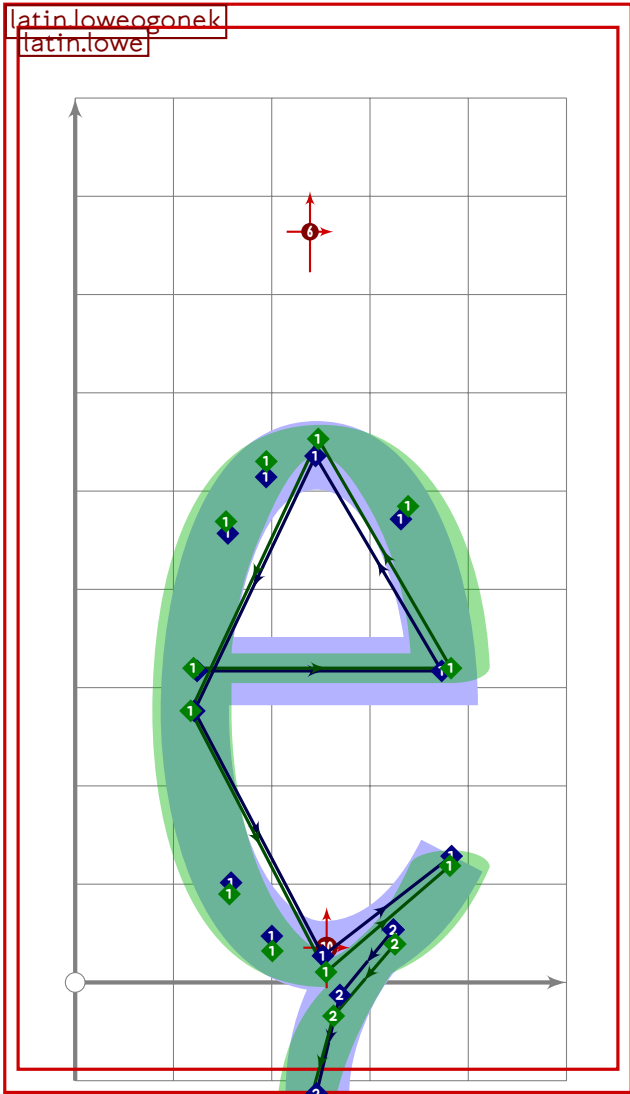
```



```

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

```



```

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

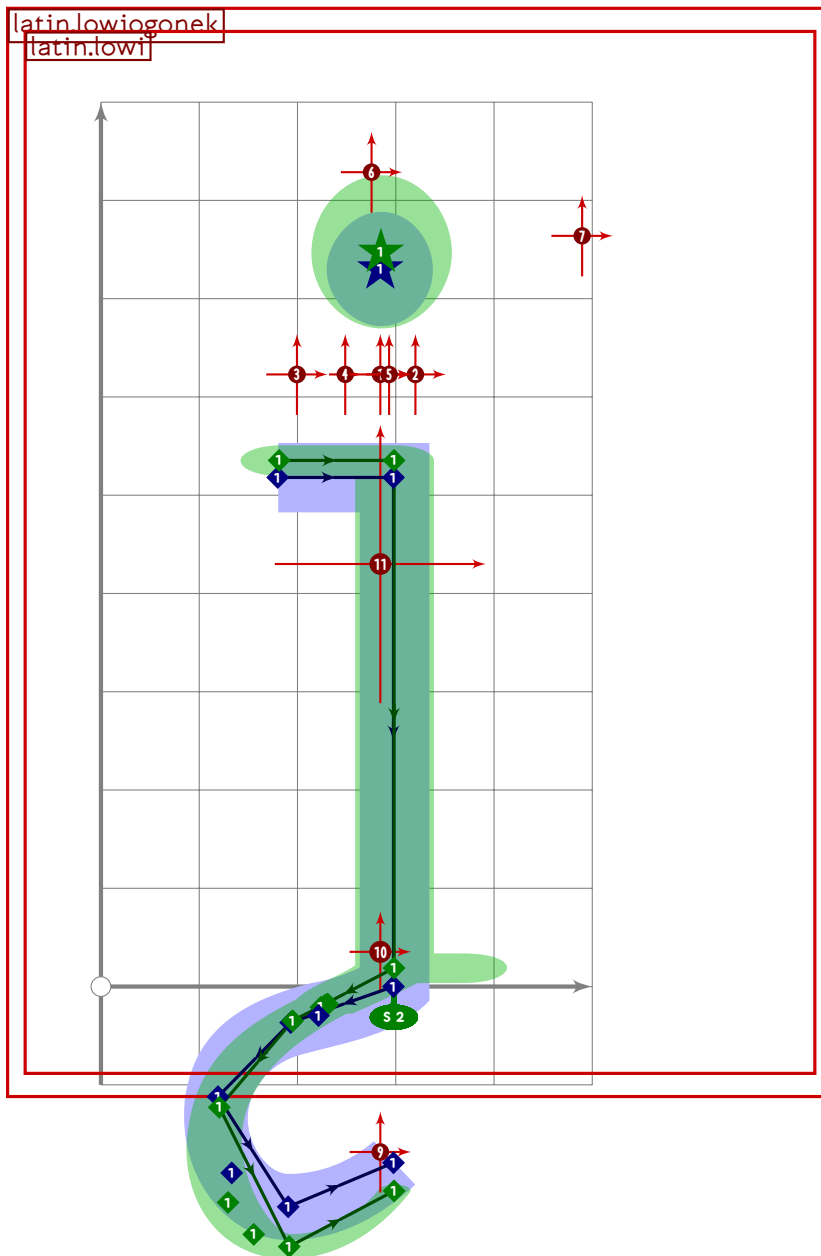
```

U+012F
tsuku.iogonek

```

159
160 push_stroke(z7{-direction (-0.5+length get_stroke(0)) of get_stroke(0)}
161 ..z8..z9{right}..z10,
162 (1,1,1)-(1,4,4)-(1,3,1.3)-(1,4,4)-(1,1));
163 replace_stroke(0,insert_nodes(oldp)(length(oldp)-2.5));
164 expand_pbox;
165 enddef;

```



```

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

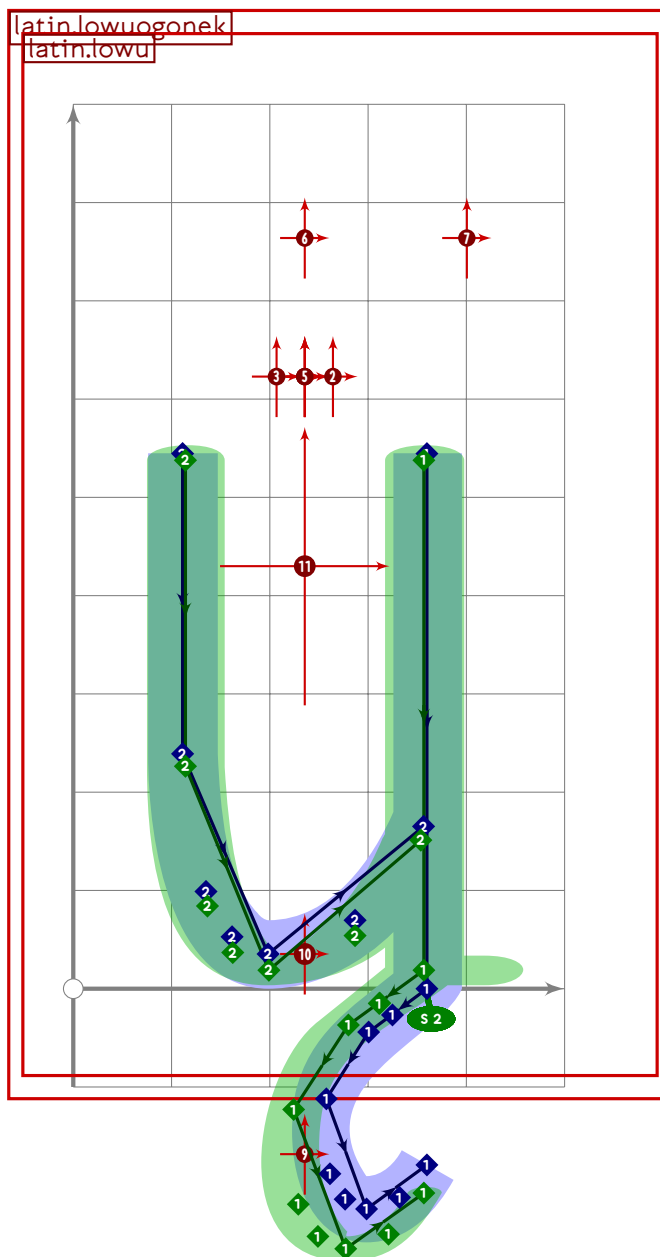
```

OGON

```

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

```



```

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

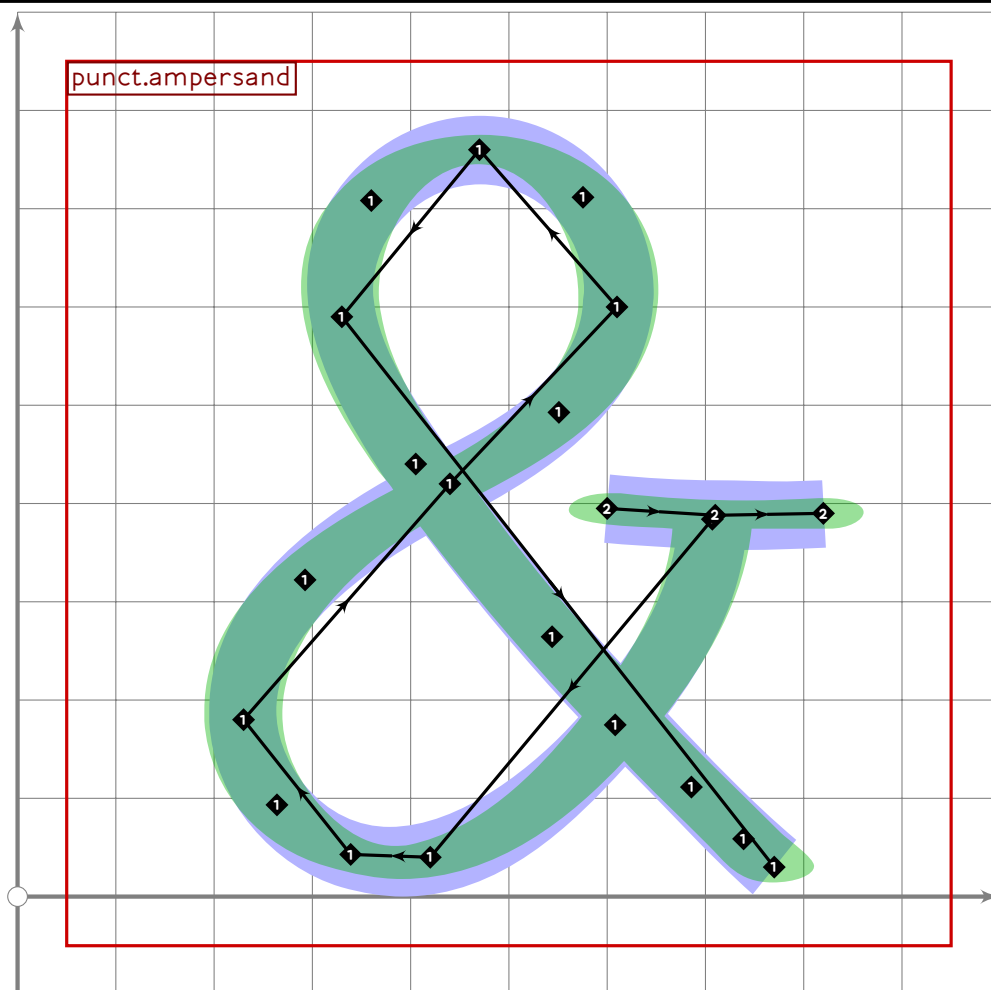
```

punct.mp

```

1 %
2 % Punctuation for Tsukurimashou
3 % Copyright (C) 2011, 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

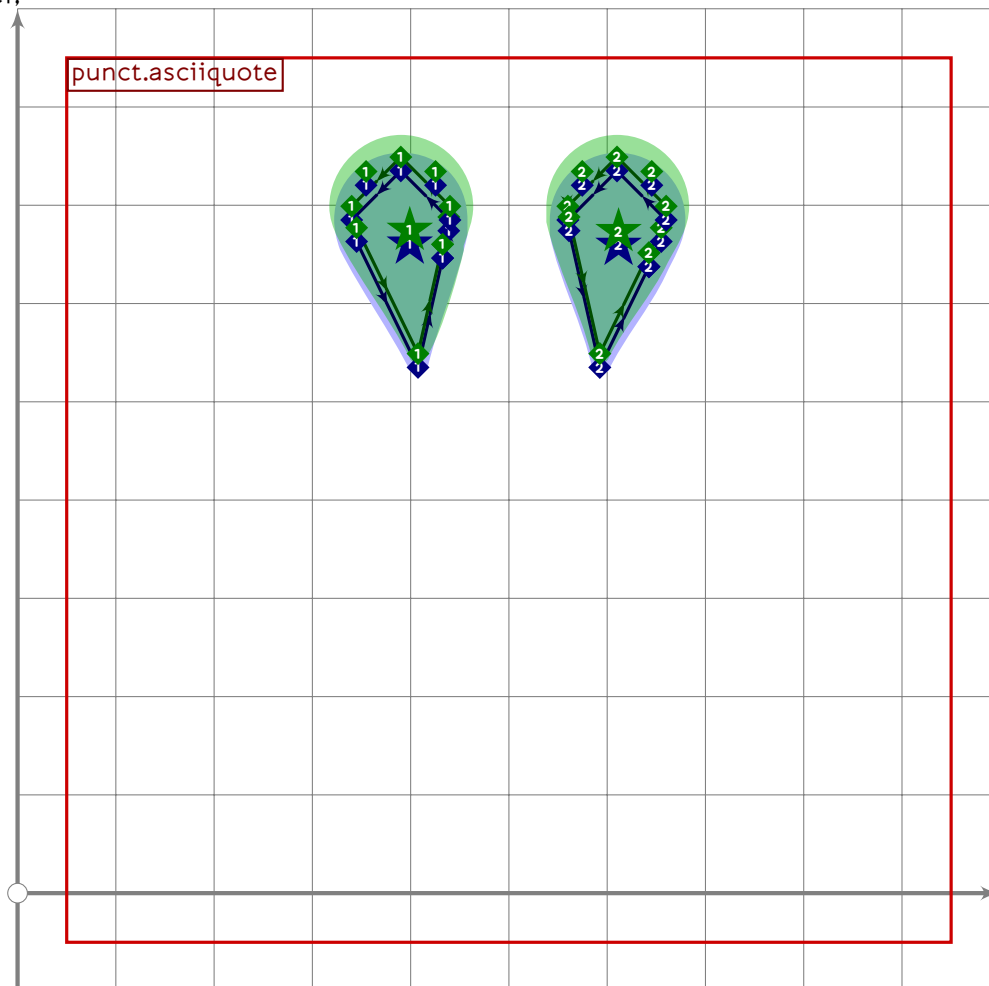
```


U+FF02
tsuku.uniFF02

```

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

```



```

48
49  vardef punct.asciiquote =
50    push_pbox_toexpand("punct.asciiquote");
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));

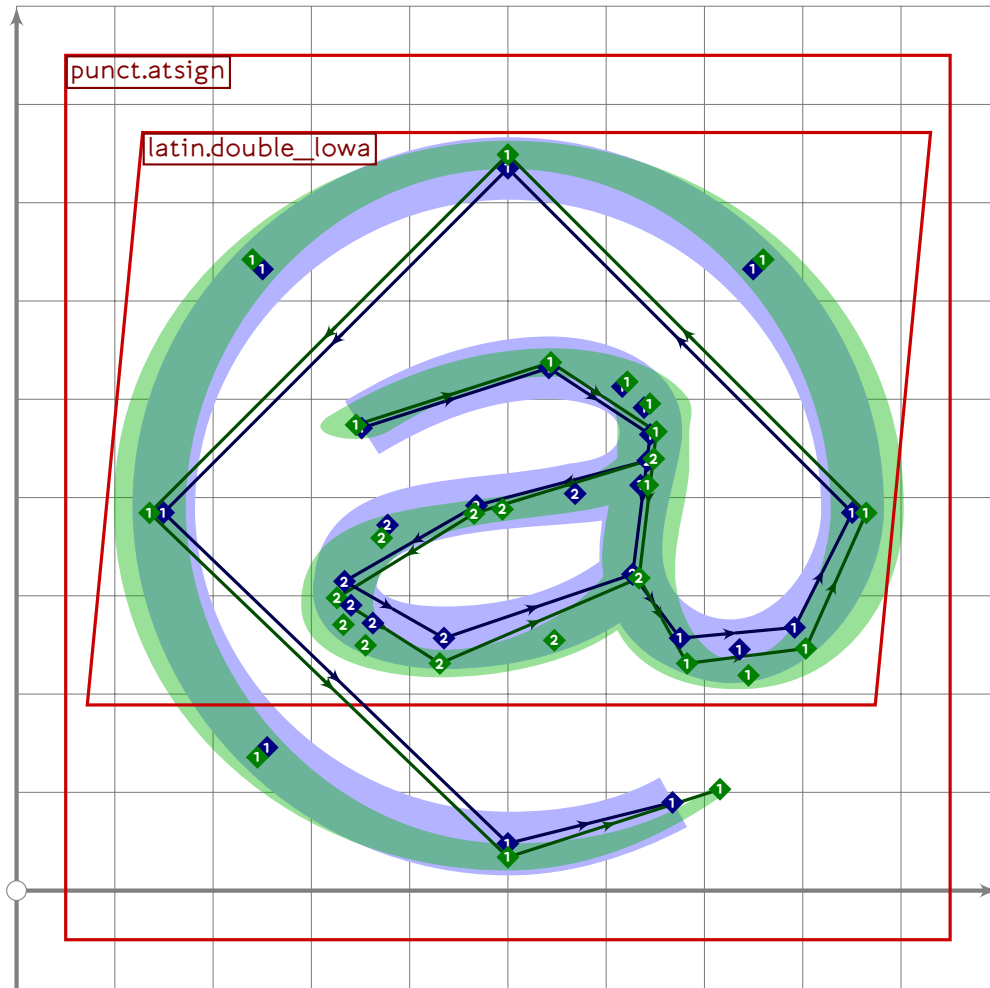
```

PUNC

```

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;

```



```

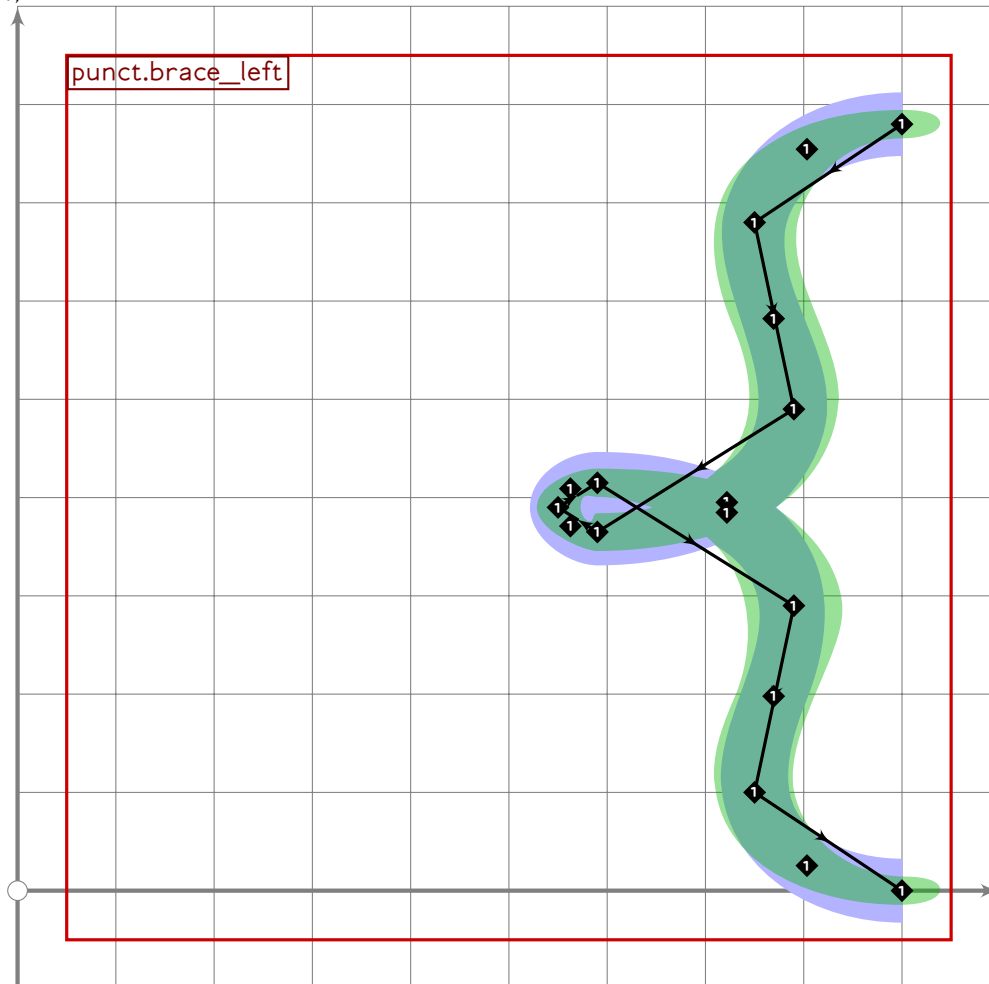
92
93 vardef punct.atsign =
94   push_pbox_toexpand("punct.atsign");
95   begingroup
96     save xsp,ysp;
97     xsp:=sp;
98     latin.low_a;
99     set_boserif(-1,3,whatever);
100    ysp:=sp;
101
102    numeric x[],y[];
103    x1-x2=x2-x3=y2-y1;
104    x2=x4=500;
105    y1=y3=0.49[y4,y2];
106    y2=latin_wide_high_r;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

```

```

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;

```



```

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

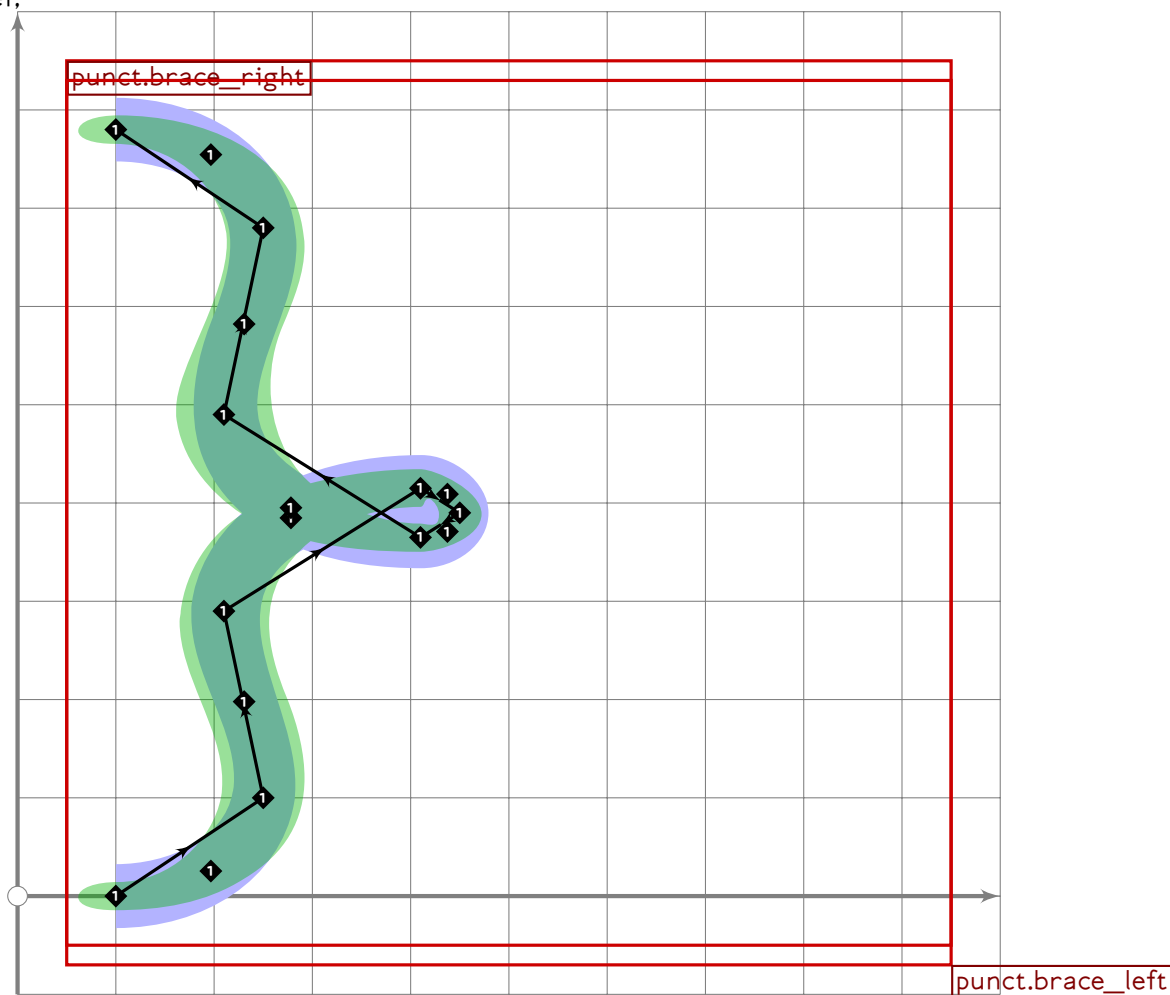
```

U+FF5D
tsuku.uniFF5D

```

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

```

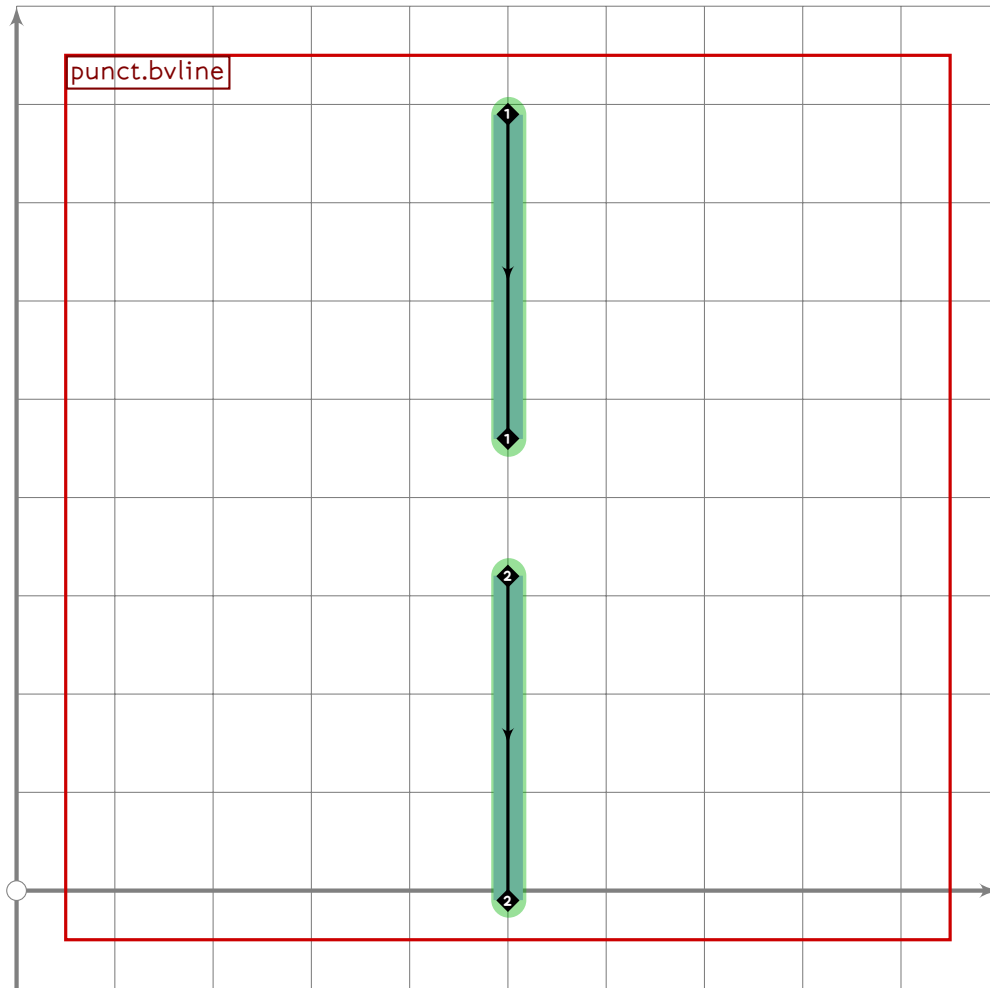


```

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

```

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_stroke(0)((point 4.2 of oldp)-oldp);
193 (0,0) transformed u=llcorner get_stroke(0);
194 (1,0) transformed u=lrcorner get_stroke(0);
195 (0,1) transformed u=ulcorner get_stroke(0);
196 replace_stroke(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_stroke(0)-cycle);
206 else:
207   replace_stroke(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;

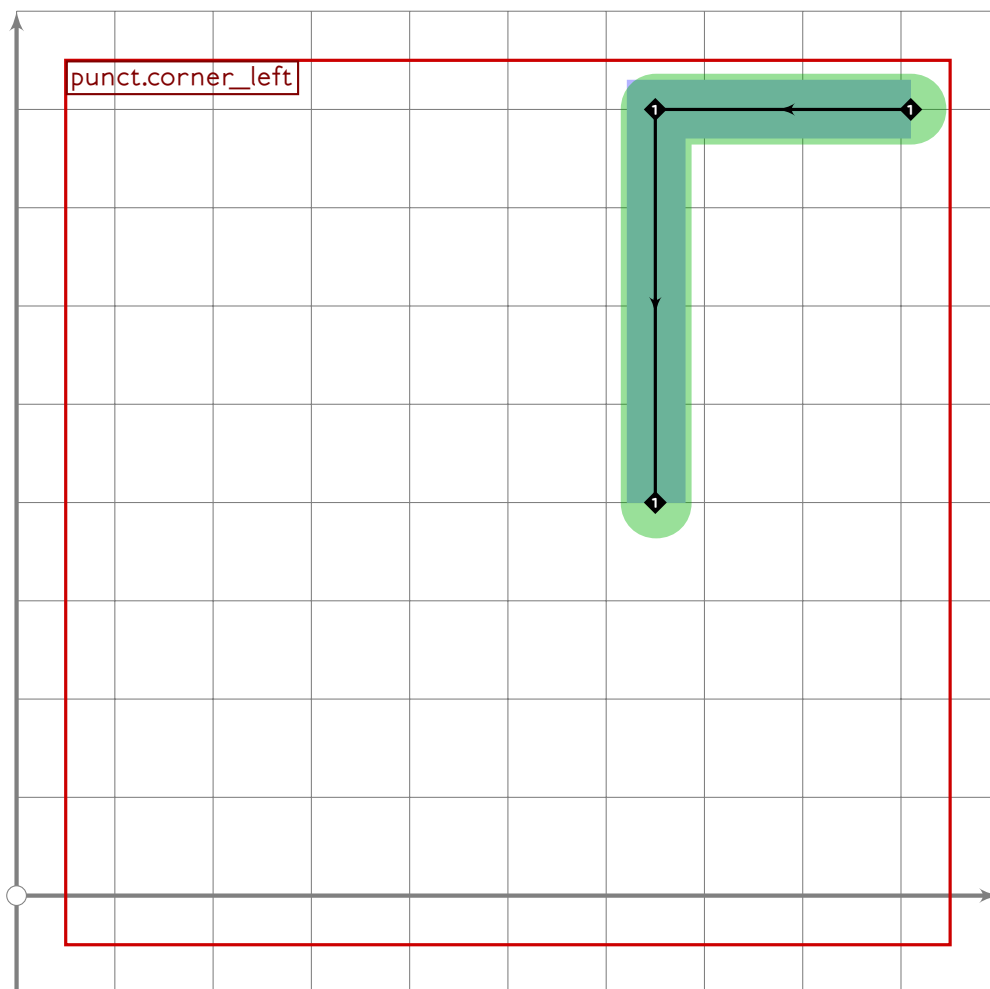
```

```

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 (cpo 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;

```

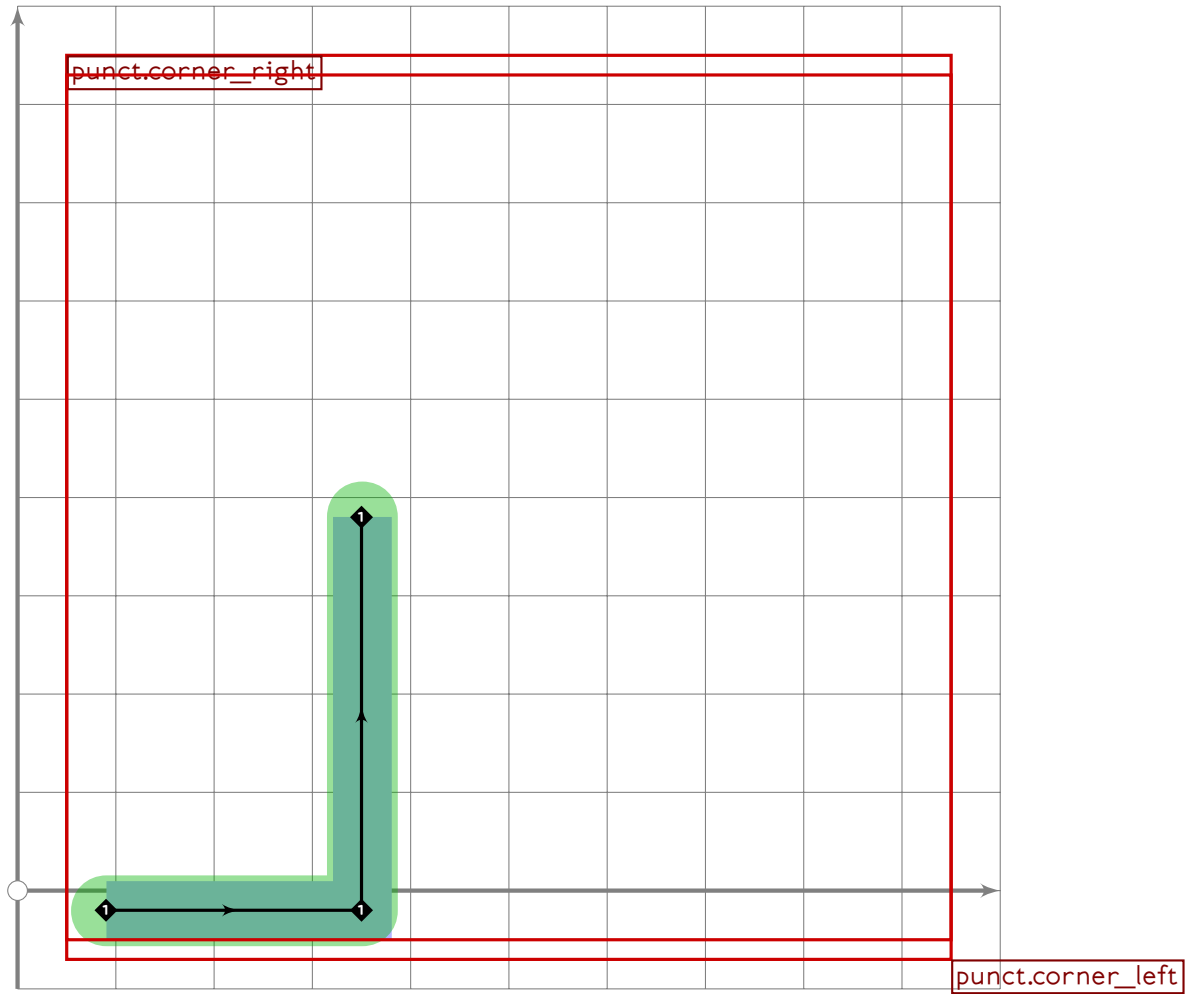

U+300C
tsuku.uni300C



```

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;

```

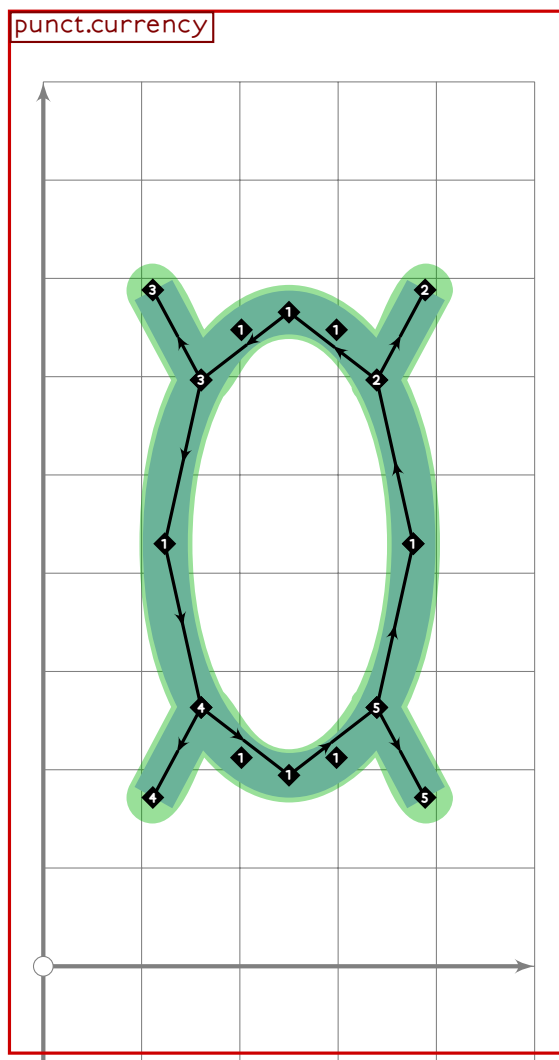


```

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;

```

U+00A4
tsuku.currency



```

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);

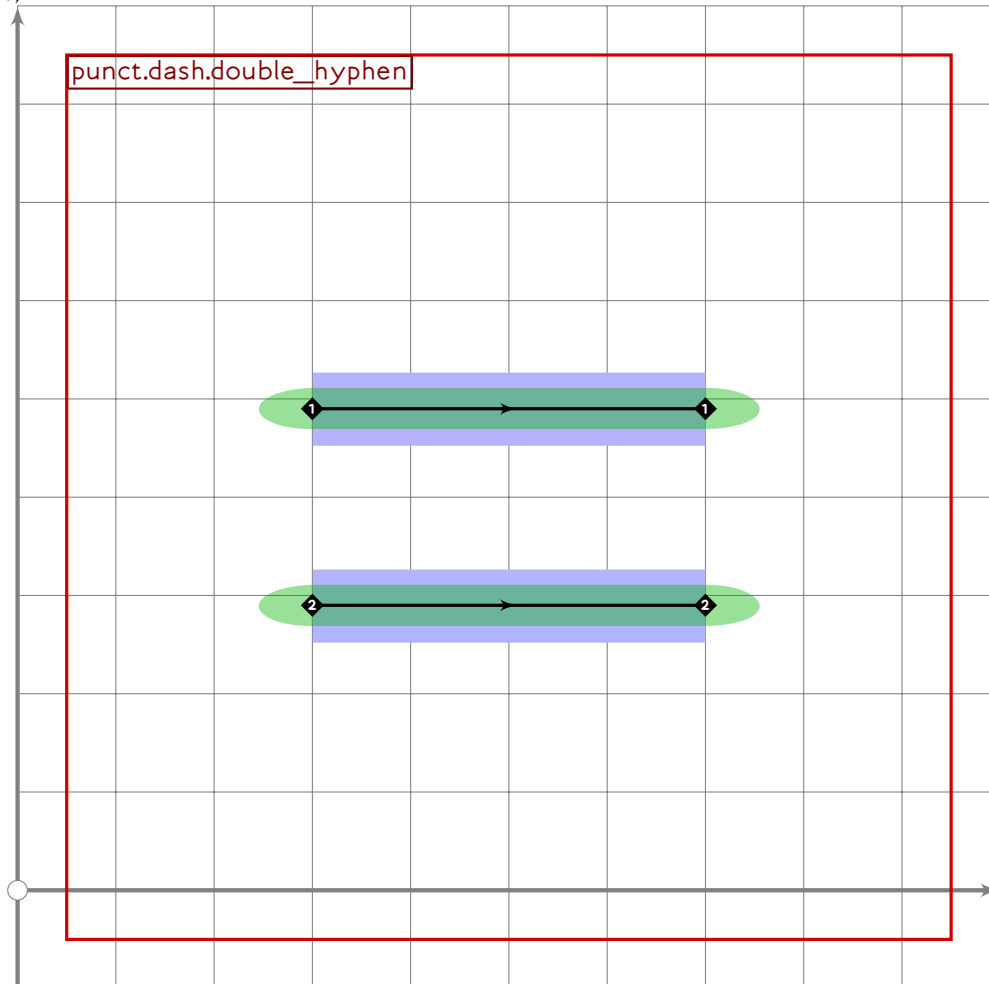
```

PUNC

```

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;

```



```

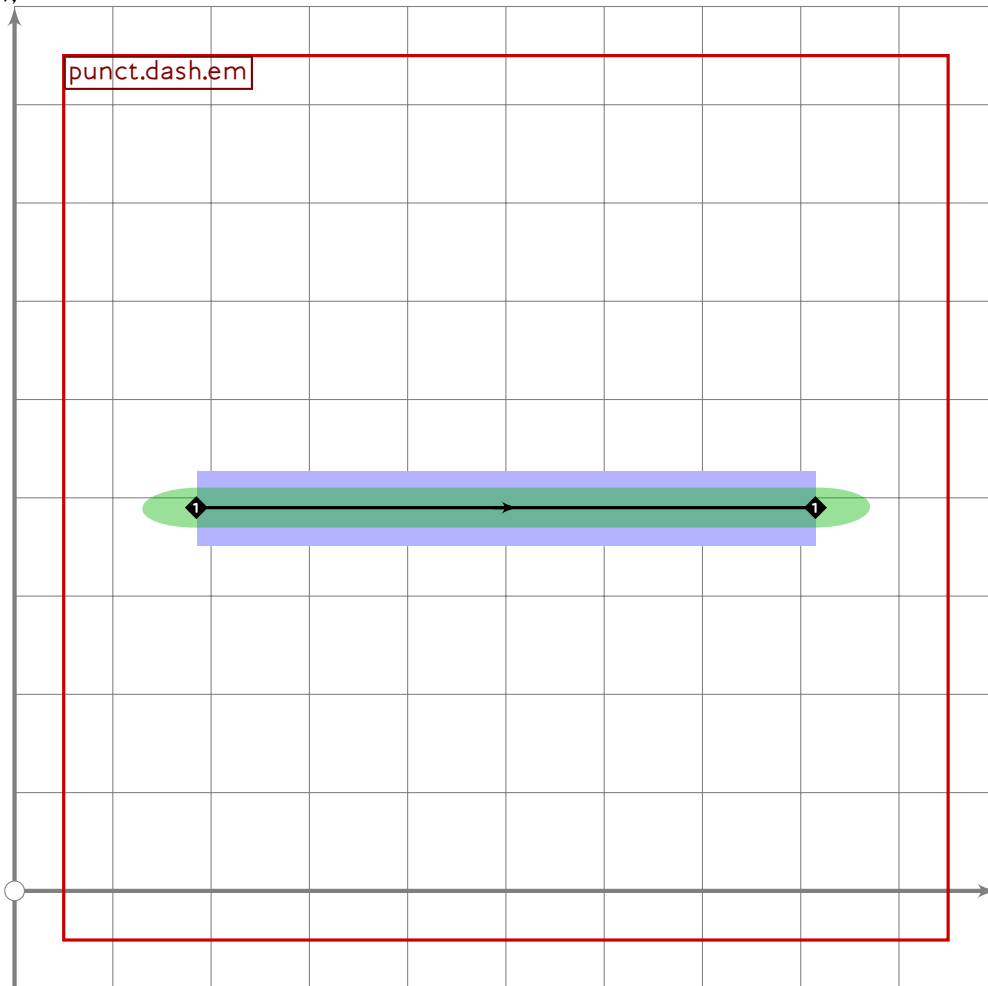
307
308 vardef punct.dash.double_hyphen =
309     push_pbox_toexpand("punct.dash.double_hyphen");
310
311     (z1+z4)/2=centre_pt;
312     x2-x1=400;

```

U+2014

tsuku.emdash

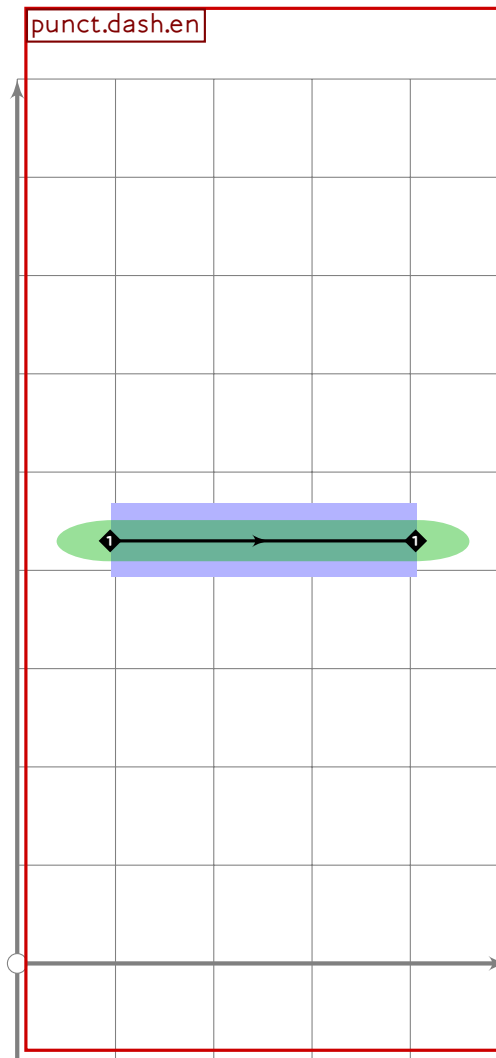
```
313 y1-y3=200;
314 y1=y2;
315 y3=y4;
316 x1=x3;
317 x2=x4;
318
319 push_stroke(z1-z2,(2,2)-(2,2));
320 % set_bobrush(0,brpunct);
321 push_stroke(z3-z4,(2,2)-(2,2));
322 % set_bobrush(0,brpunct);
323
324 expand_pbox;
325 enddef;
```



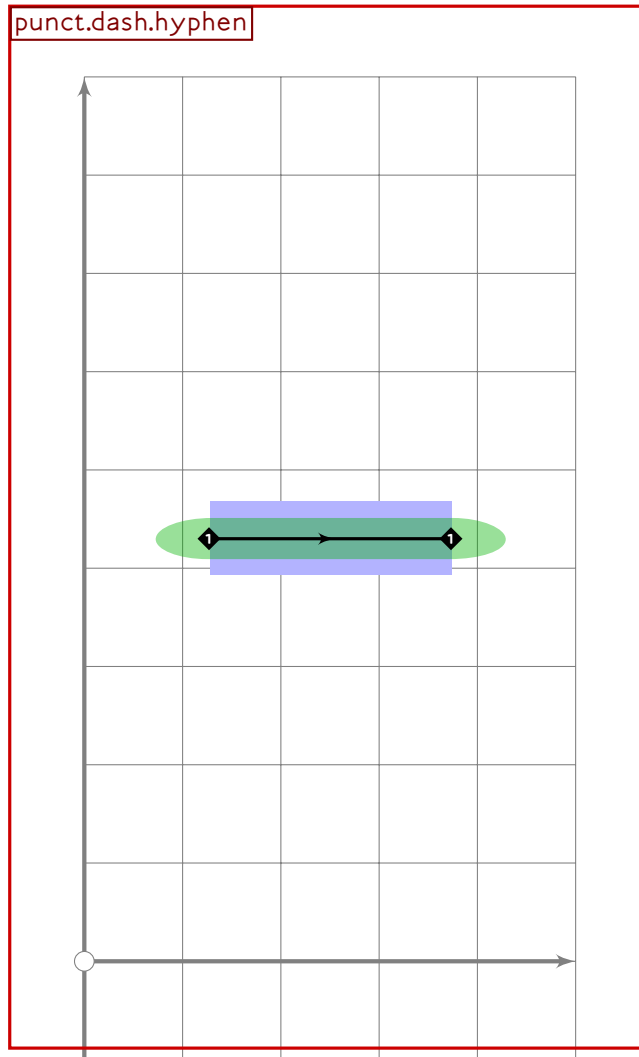
```
326
327 vardef punct.dash.em =
328   push_pbox_toexpand("punct.dash.em");
329   (z1+z2)/2=centre_pt;
330   x2-x1=if is_proportional: 750 else: 630 fi;
331   y1=y2;
332   push_stroke(z1-z2,(2,2)-(2,2));
333   % set_bobrush(0,brpunct);
```

PUNC

```
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;
```



```

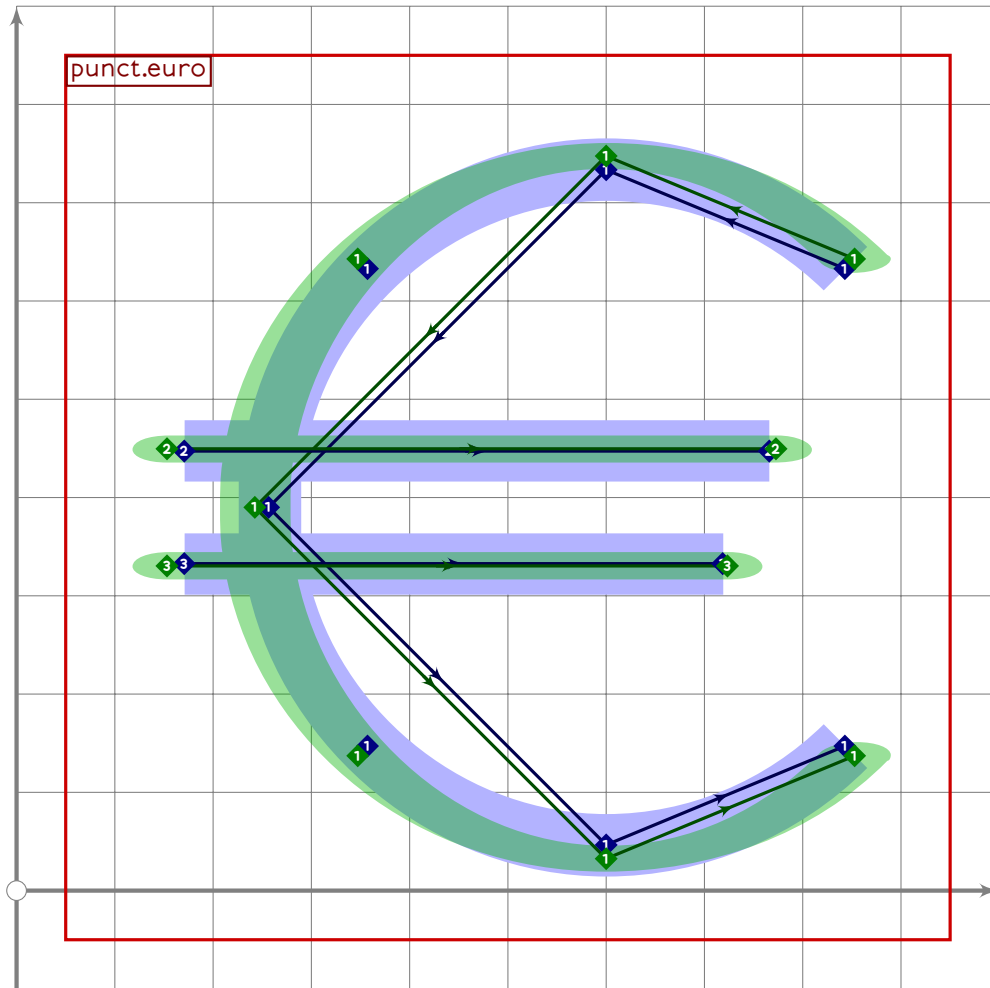
346
347 vardef punct.dash.hyphen =
348   push_pbox_toexpand("punct.dash.hyphen");
349   (z1+z2)/2=centre_pt;
350   x2-x1=340;
351   y1=y2;
352   push_stroke(z1-z2,(2,2)-(2,2));
353   % set_bobrush(0,brpunct);
354   expand_pbox;
355 enddef;
356
357 vardef punct.dash.long =
358   push_pbox_toexpand("punct.dash.long");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

```

```

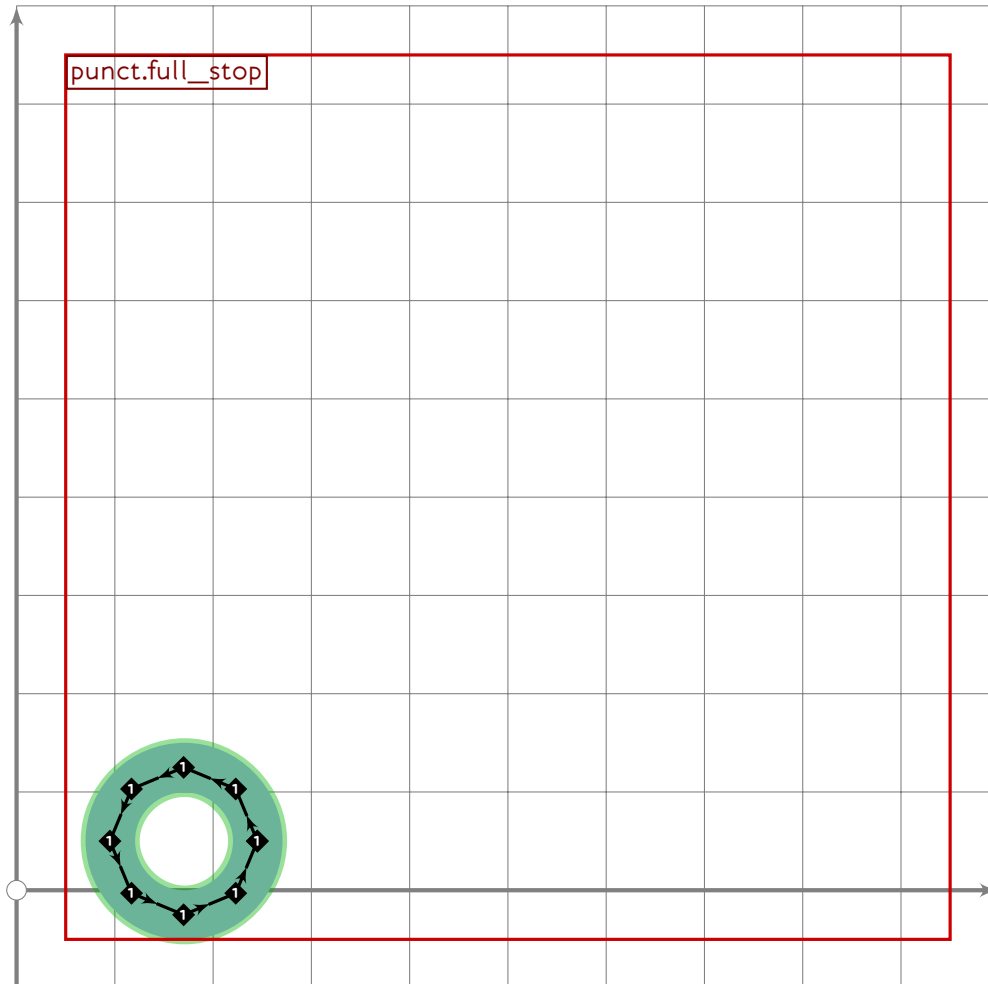
393
394 vardef punct.euro =
395   push_pbox_toexpand("punct.euro");
396
397   push_stroke((subpath (0.5,3.5) of ((1,0)..(0,1)..(-1,0)..(0,-1)..cycle))
398     scaled ((latin_wide_high_r-latin_wide_low_r)/2)
399     shifted (centre_pt+(100,0)),
400     (1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6)-(1.6,1.6));
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)),

```

```

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

```



```

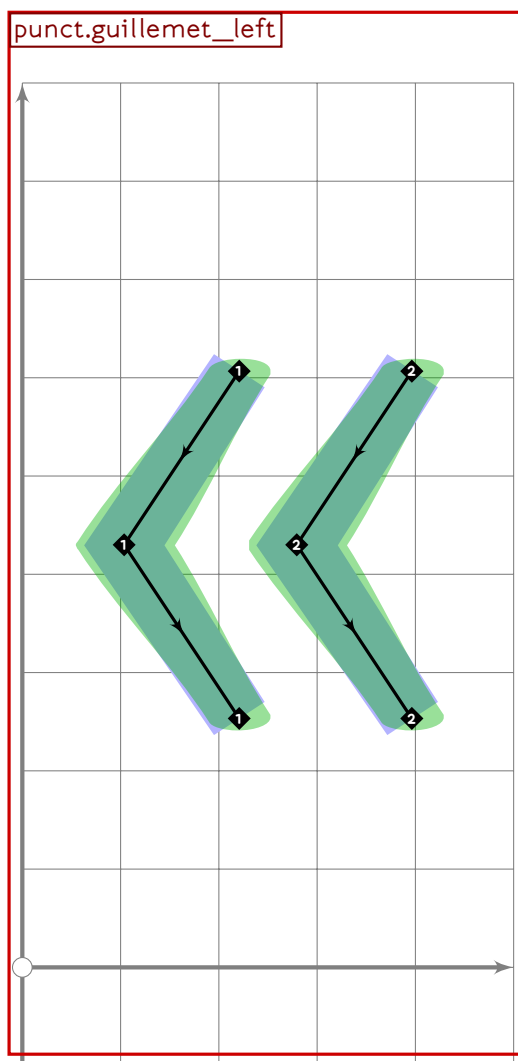
419 % this is *ideographic* full stop, not Latin period
420 vardef punct.full_stop =
421   push_pbox_toexpand("punct.full_stop");
422
423   if tsu_brush_max.brpunct*100>=tsu_punct_size:
424     push_lcblob(fullcircle
425       xscaled (1.5*tsu_punct_size+tsu_brush_max.brpunct*100)
426       yscaled (1.5*tsu_punct_size+tsu_brush_max.brpunct*100
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;

```

U+00AB

tsuku.guillemotleft

```
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;
```



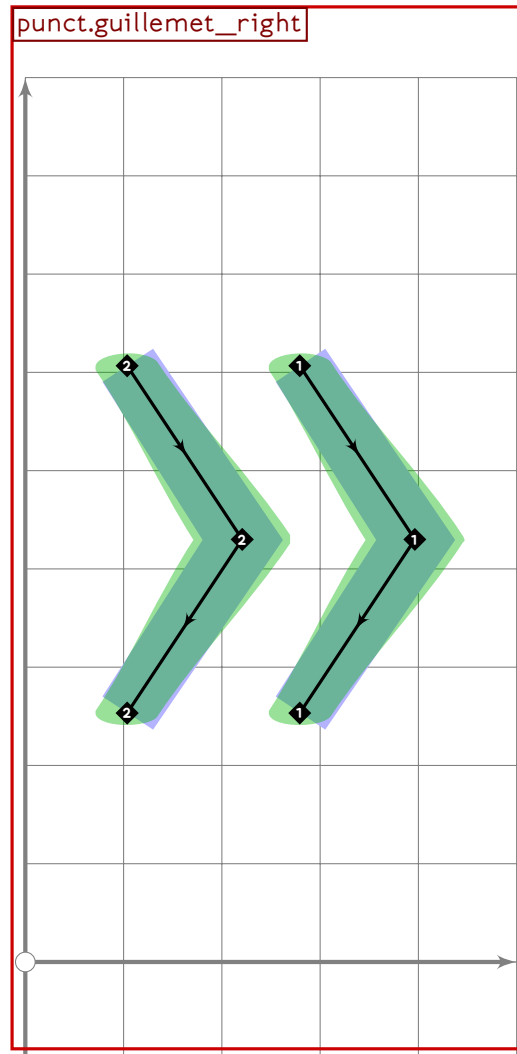
```
447
448 vardef punct.guillemet_left =
449   push_pbox_toexpand("punct.guillemet_left");
450
451   push_stroke(((−0.5,1.5)−(−2.5,0)−(−0.5,−1.5))
452     scaled tsu_punct_size shifted centre_pt,
453     (1.5,1.5)−(2,2)−(1.5,1.5));
```

PUNC

```

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;

```



```

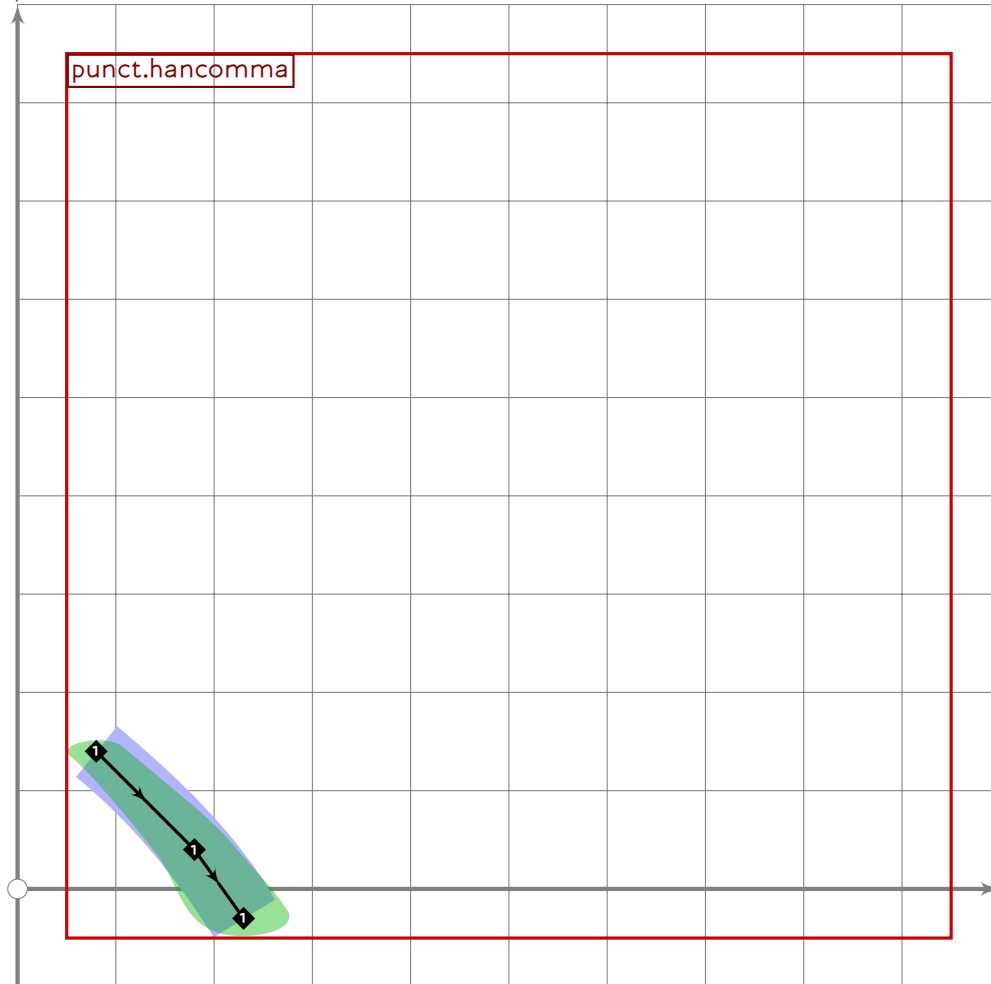
464
465 vardef punct.guillemet_right =
466     push_pbox_toexpand("punct.guillemet_right");
467
468     push_stroke(((0.5,1.5)-(2.5,0)-(0.5,-1.5))
469         scaled tsu_punct_size shifted centre_pt,
470         (1.5,1.5)-(2,2)-(1.5,1.5));
471     set_bosize(0,90);
472     set_botip(0,1,1);

```

```

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;

```



```

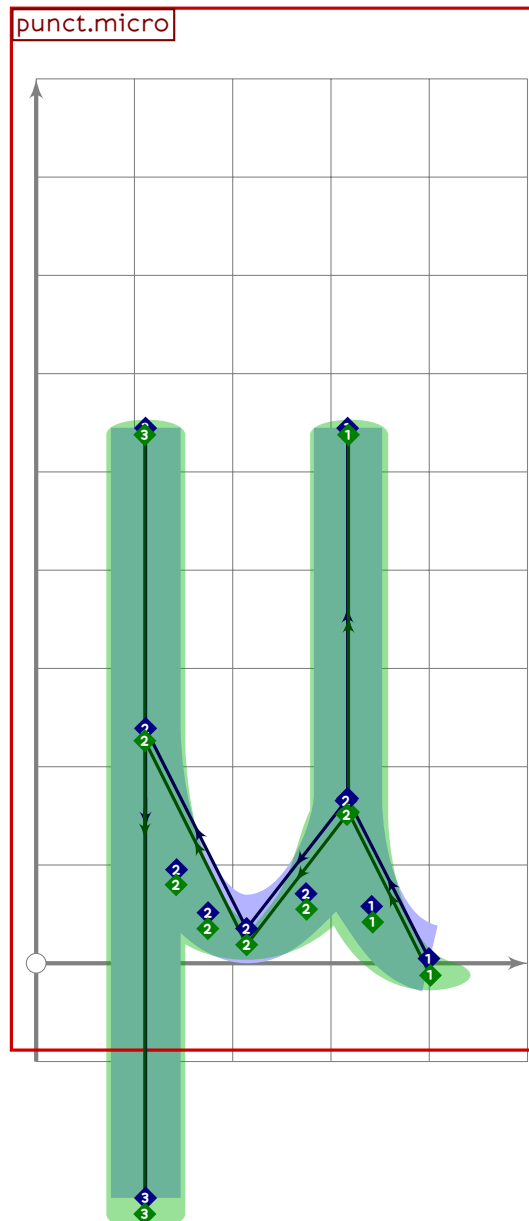
481
482 vardef punct.hancomma =
483   push_pbox_toexpand("punct.hancomma");
484   push_stroke((80,140)..(180,40)..(230,−30),(1,3,1,3)..(1,6,1,6)..(1,8,1,8));
485   expand_pbox;
486 enddef;
487
488 vardef punct.hminus(expr t) =
489   push_stroke(((−1,0)−(1,0)) transformed t,(2,2)−(2,2));
490   % set_bobrush(0,brpunct);
491
492   push_pbox_explicit("punct.hminus",
493     identity shifted (−0.5,−0.5) xyscaled (2.4,0.6) transformed t);

```

```

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);
501
502   push_pbox_explicit("punct.less_than",
503     identity shifted (-0.5,-0.5) scaled 2.4 transformed t);
504 endif;
505

```

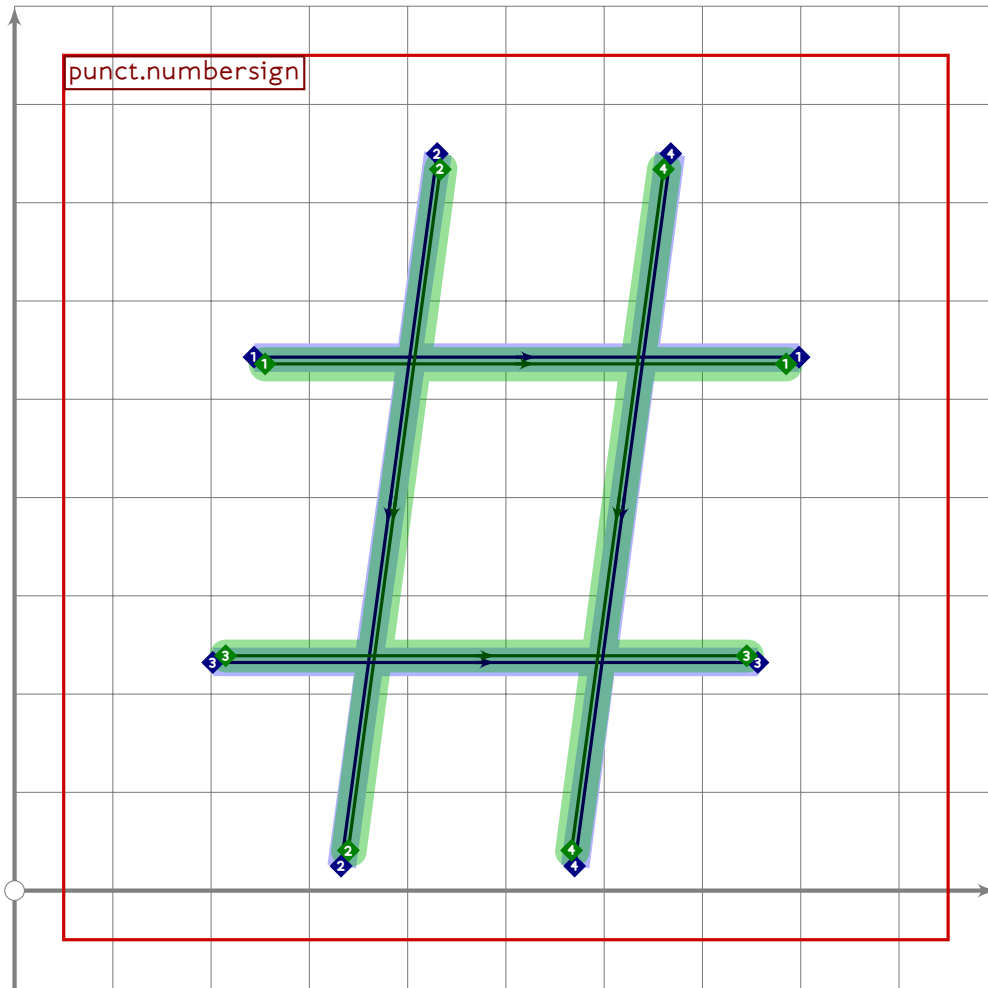


```
506 % in the future, this will probably become greek.lowmu
507 vardef punct.micro =
508   push_pbox_toexpand("punct.micro");
```

```

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 2.4 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);

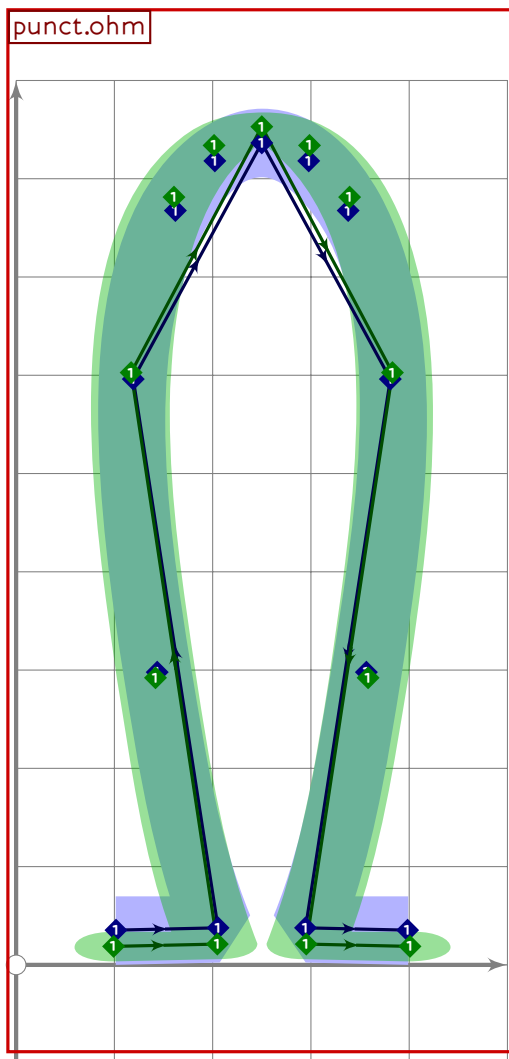
```


U+2126
tsuku.uni2126

```

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

```



```

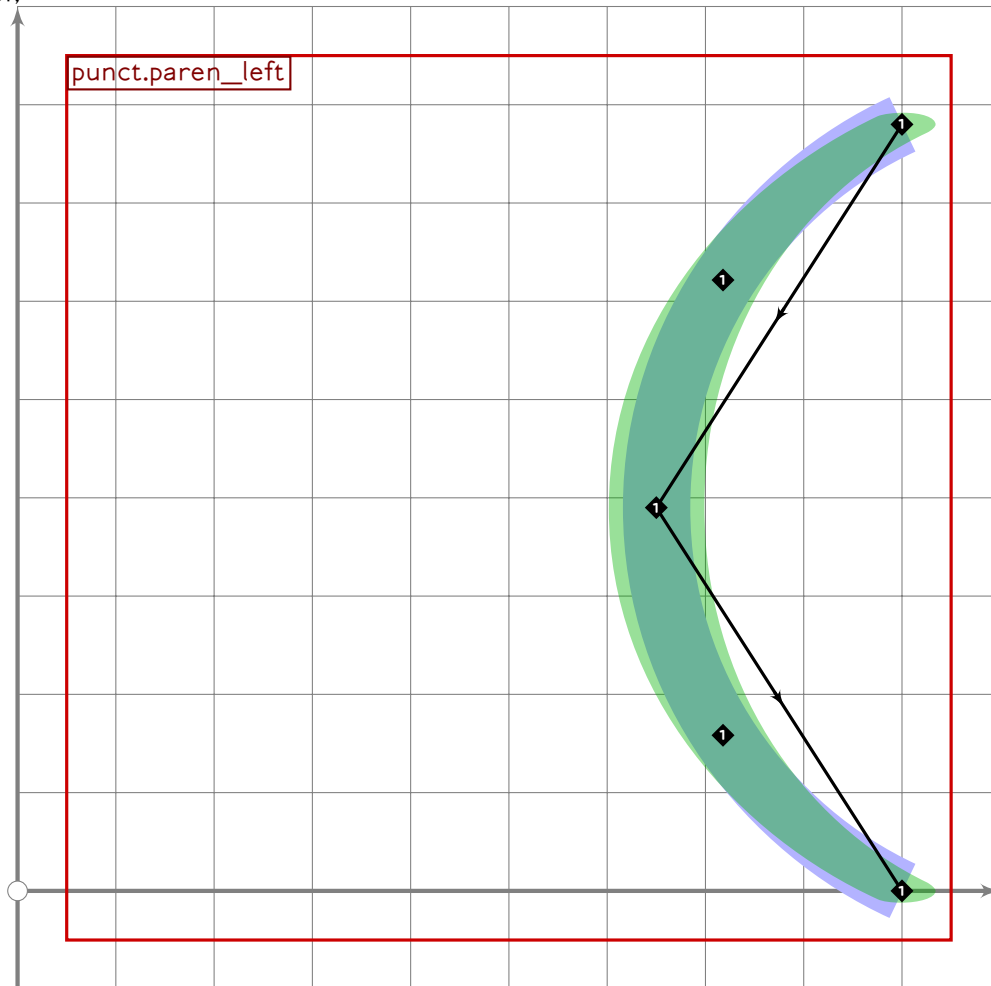
572 % in the future, this will probably become greek.upomega
573 vardef punct.ohm =
574   push_pbox_toexpand("punct.ohm");
575
576   (x5+x3)/2=(x6+x2)/2=(x7+x1)/2=x4=500;
577   x2=0.7[x1,x4];
578   x7-x1=0.76*(y4-y1);
579   x5-x3=0.67*(y4-y1);
580

```

```

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 endif;

```

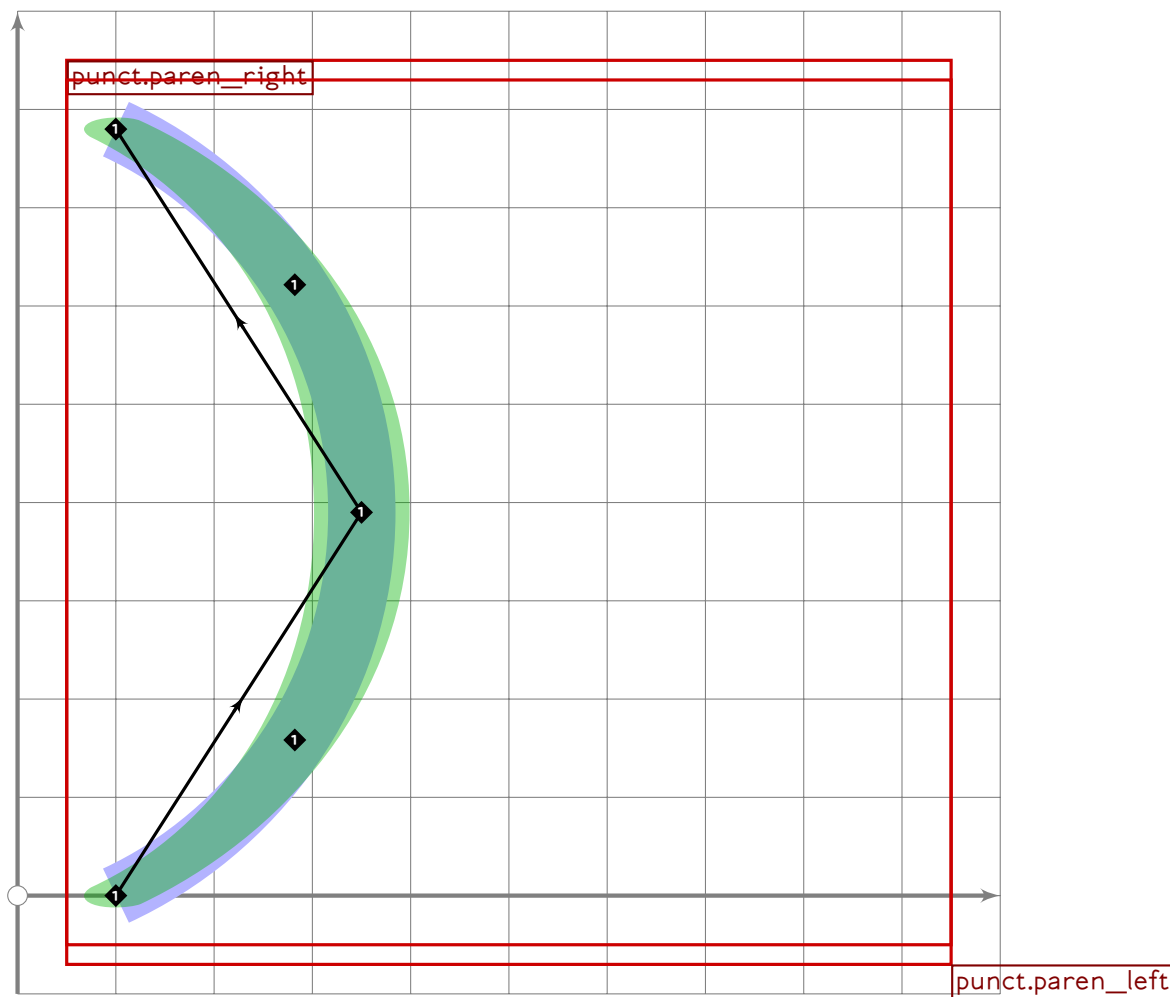


```

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 endif;

```

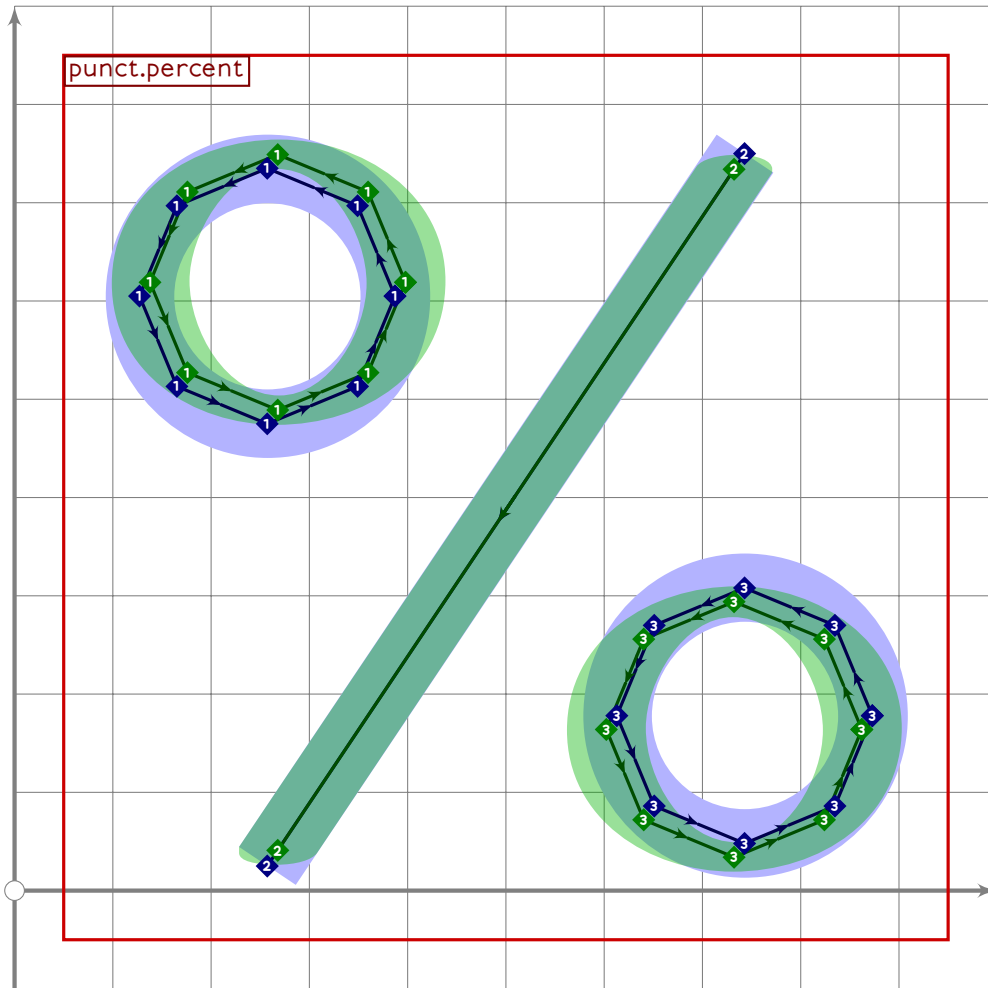
U+FF09
tsuku.uniFF09



```

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;

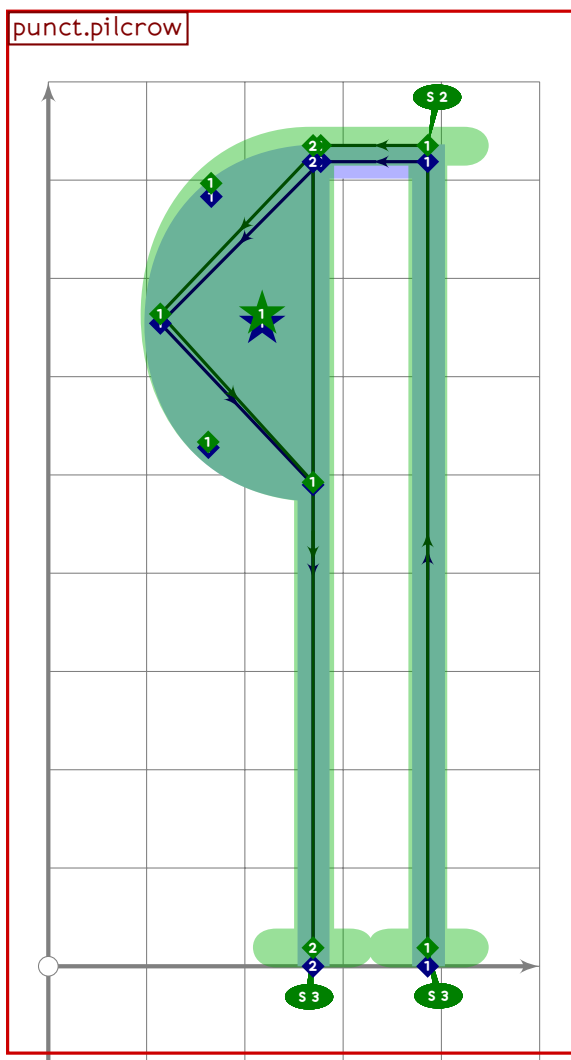
```



```

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;

```



```

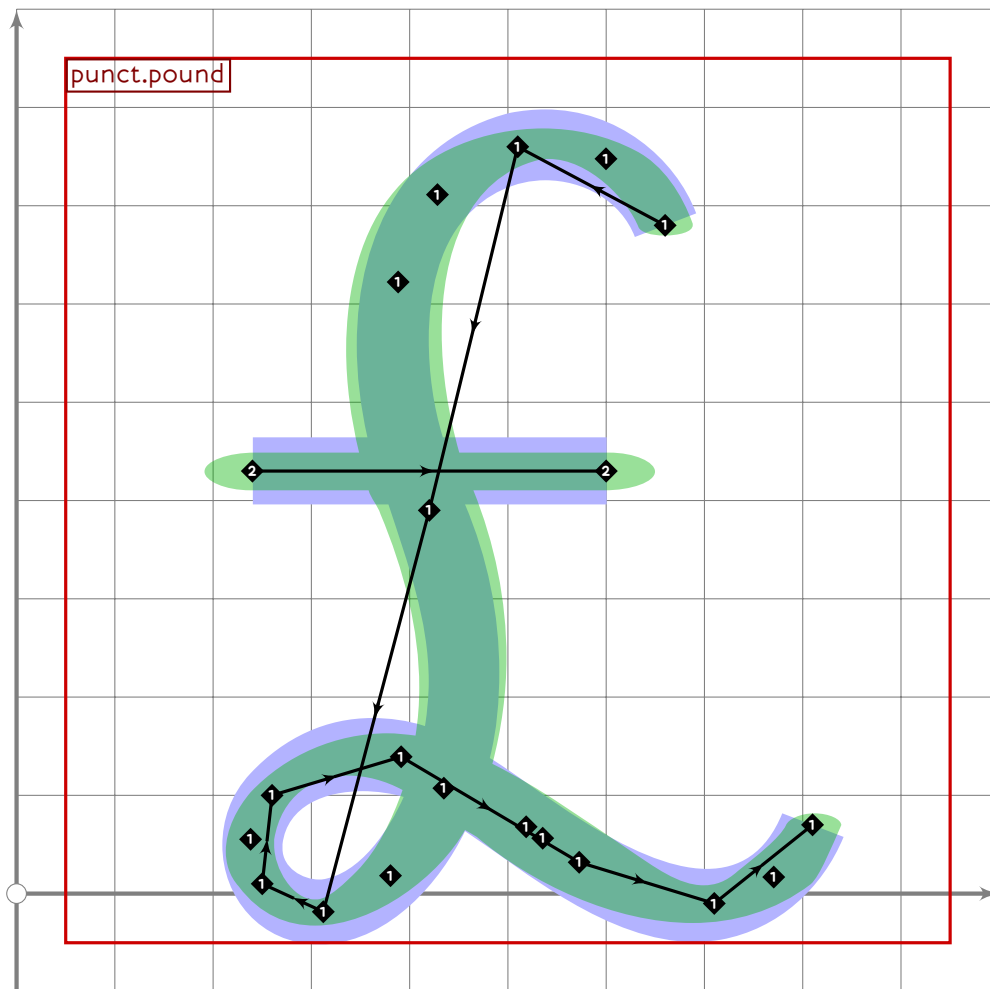
630
631 vardef punct.pilcrow =
632   push_pbox_toexpand("punct.pilcrow");
633
634   x1=x2=x6=710;
635   x3=x5=x1-420*0.4;
636   x4=x1-420;
637
638   y1=latin_wide_low_v;
639   y2=y3=latin_wide_high_h;
640   y4=(y3+y5)/2;
641   y5=y6=vmetric(0.58);
642
643   x7=x8=x9=x1-1.8*tsu_punct_size;
644   y7=y2+50;
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_strokep(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(0,1,1);
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;

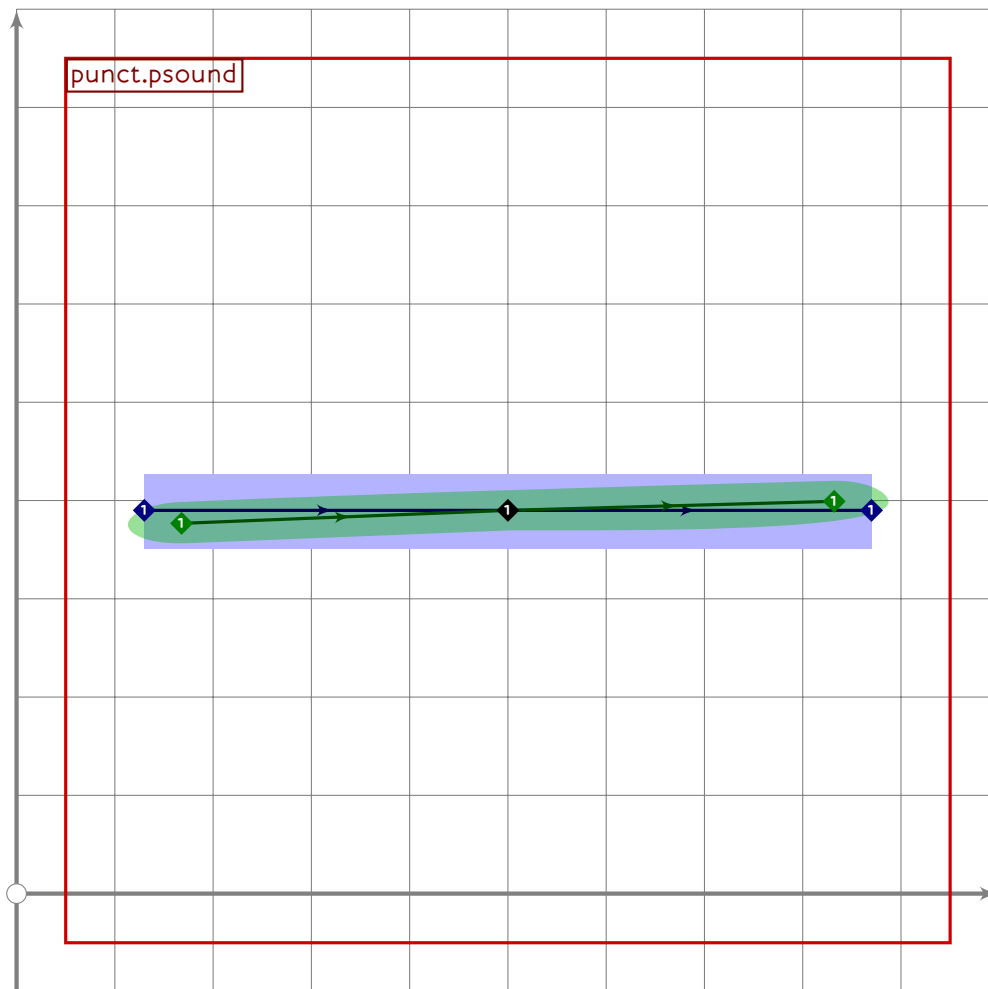
```



```

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)(2.8,4.3,4.7));
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",

```


tsuku.section



729

```
731 push_pbox_toexpand("punct.section");
```

732

```
734 x5=0.8[x2,x1];
```

—

```
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 $y_4 - y_5 = y_5 - y_6;$

742

```
743 push_stroke((z1..z2..z3..z4..z5..z6) rotatedaround (centre_pt,6),
```

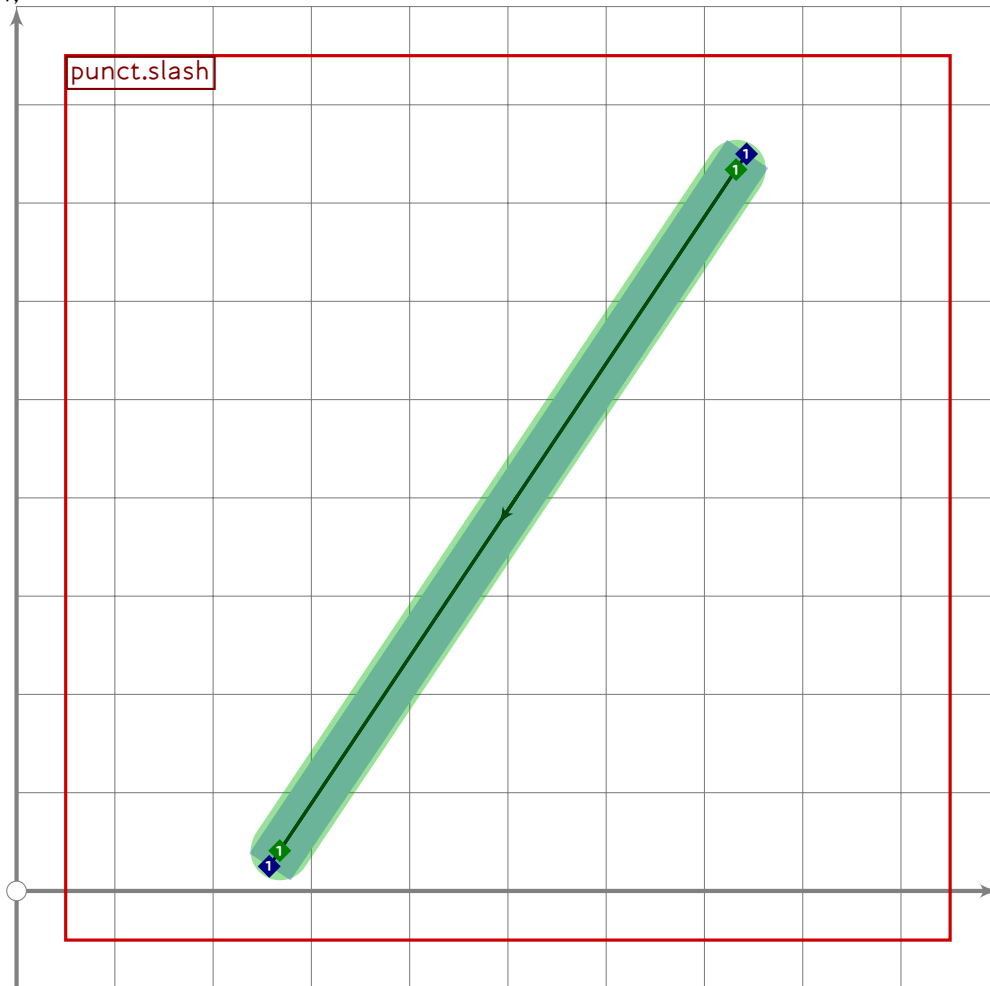
744 $(1.8,1.8)-(1.2,1.2)-(1.7,1.7)-(1.3,1.3)-(2,2)-(1.5,1.5));$

745

```

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

```

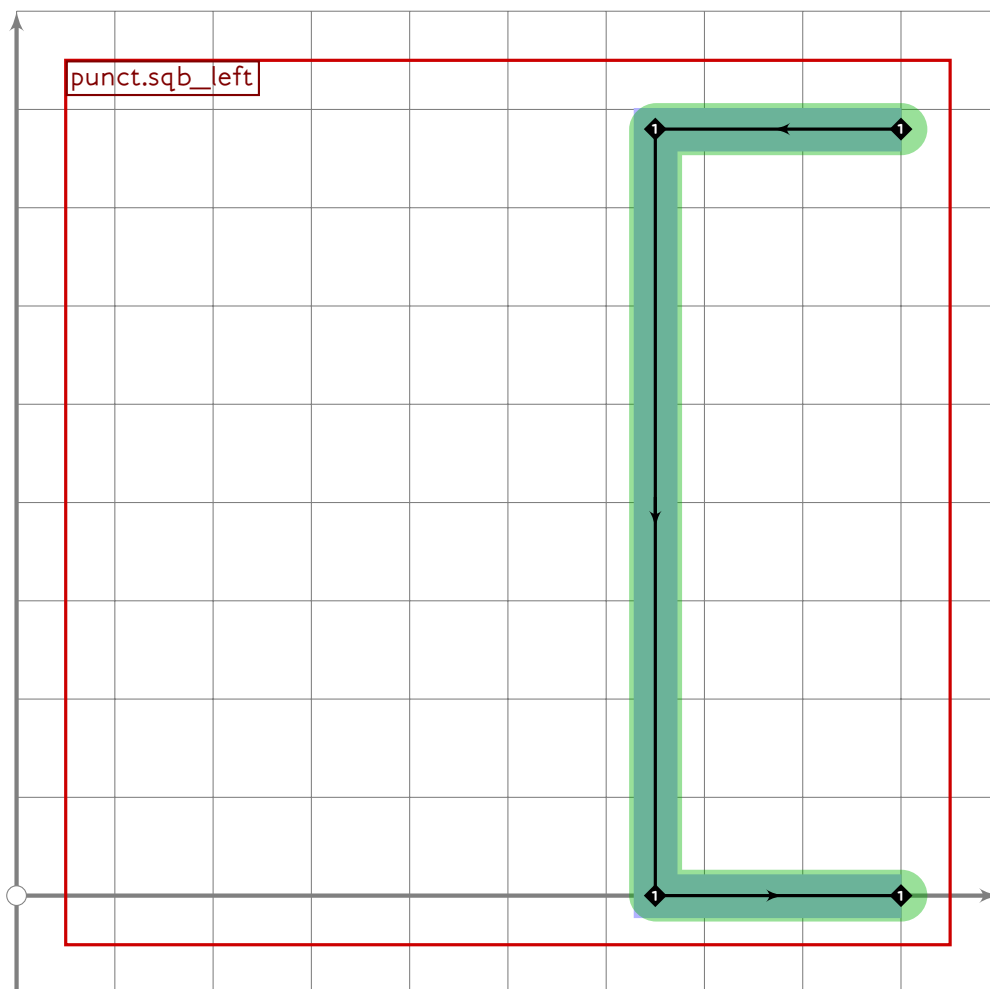


```

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;

```

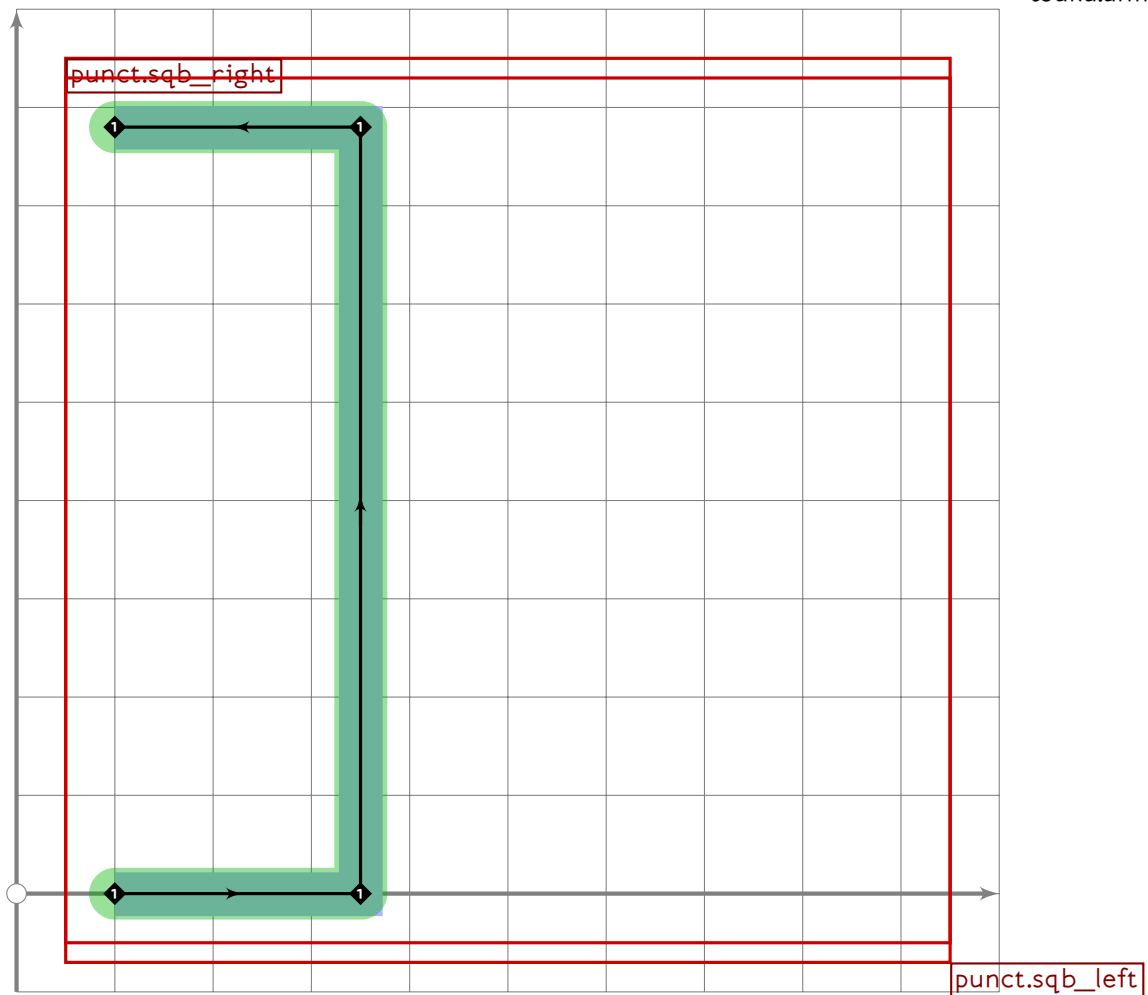
U+FF3B
tsuku.uniFF3B



```

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;

```

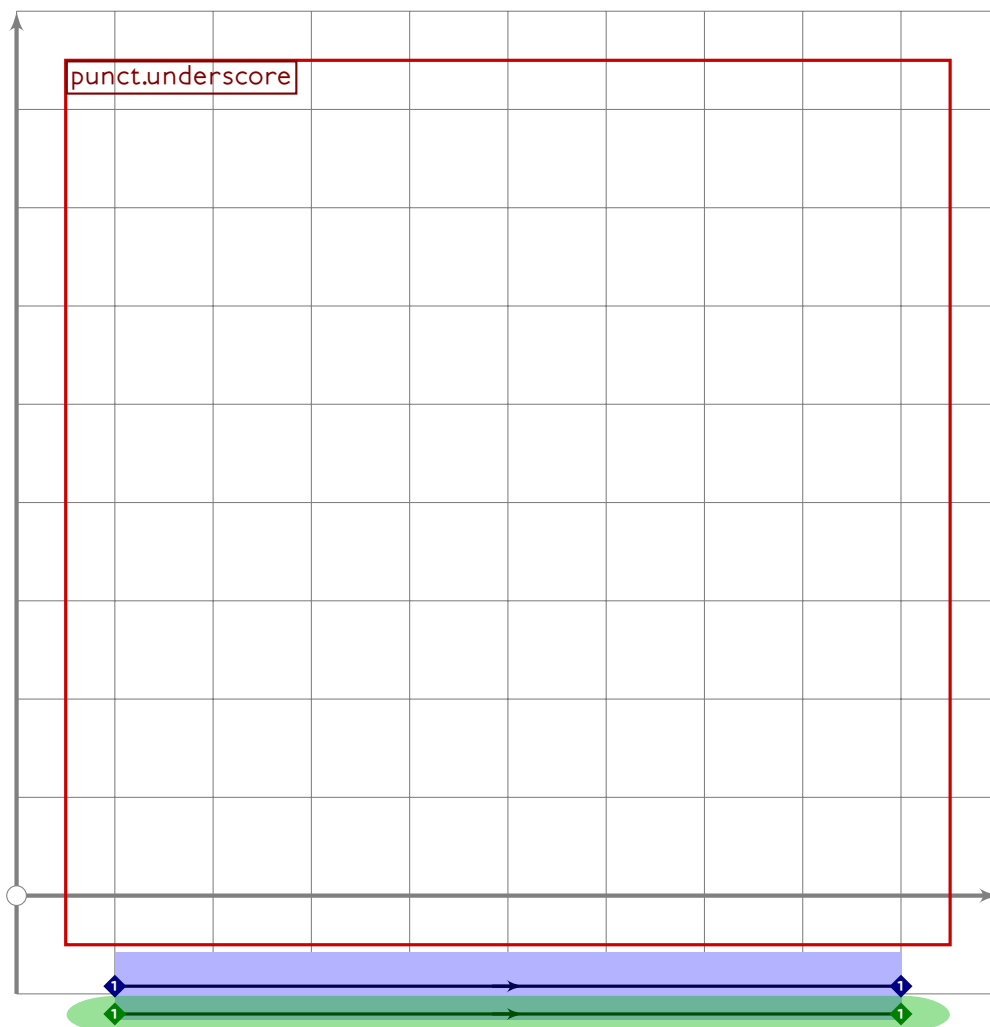


```

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 2.4 transformed t);
796 enddef;

```

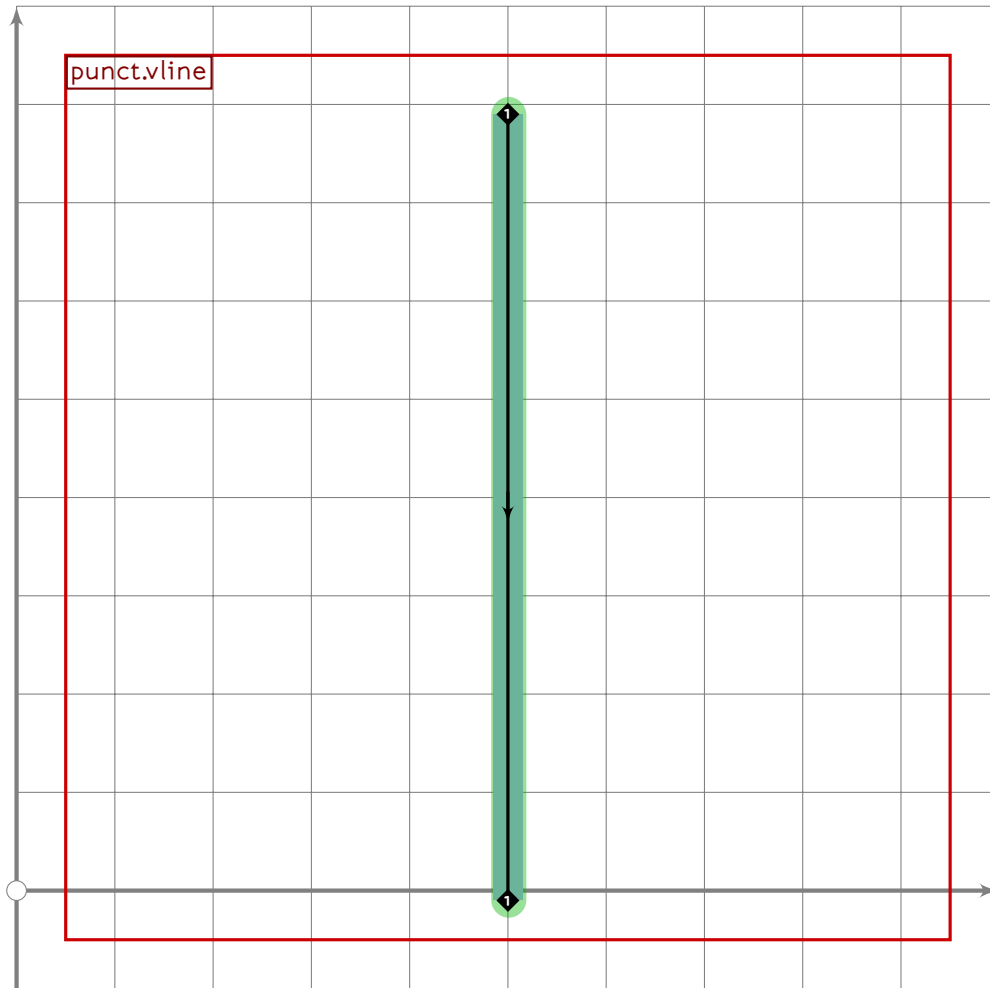
U+FF3F
tsuku.uniFF3F



```

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;

```

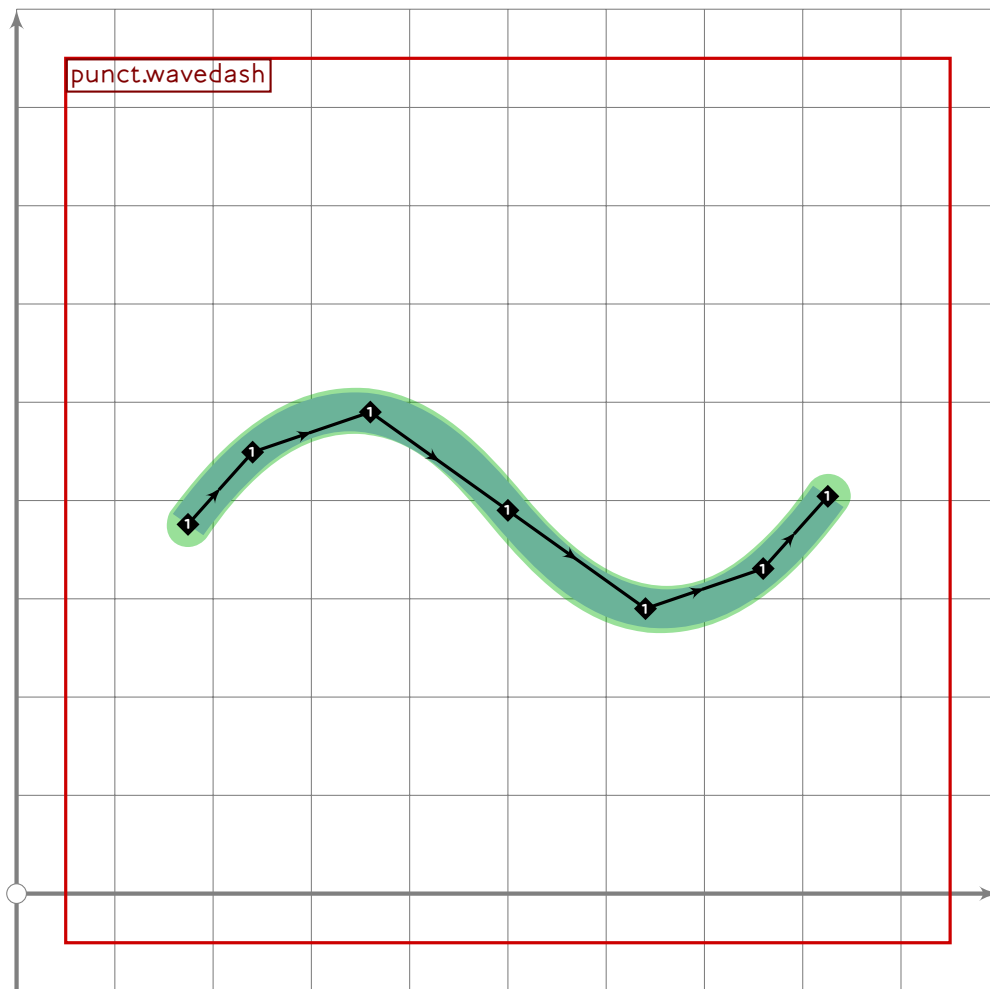


```

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;

```

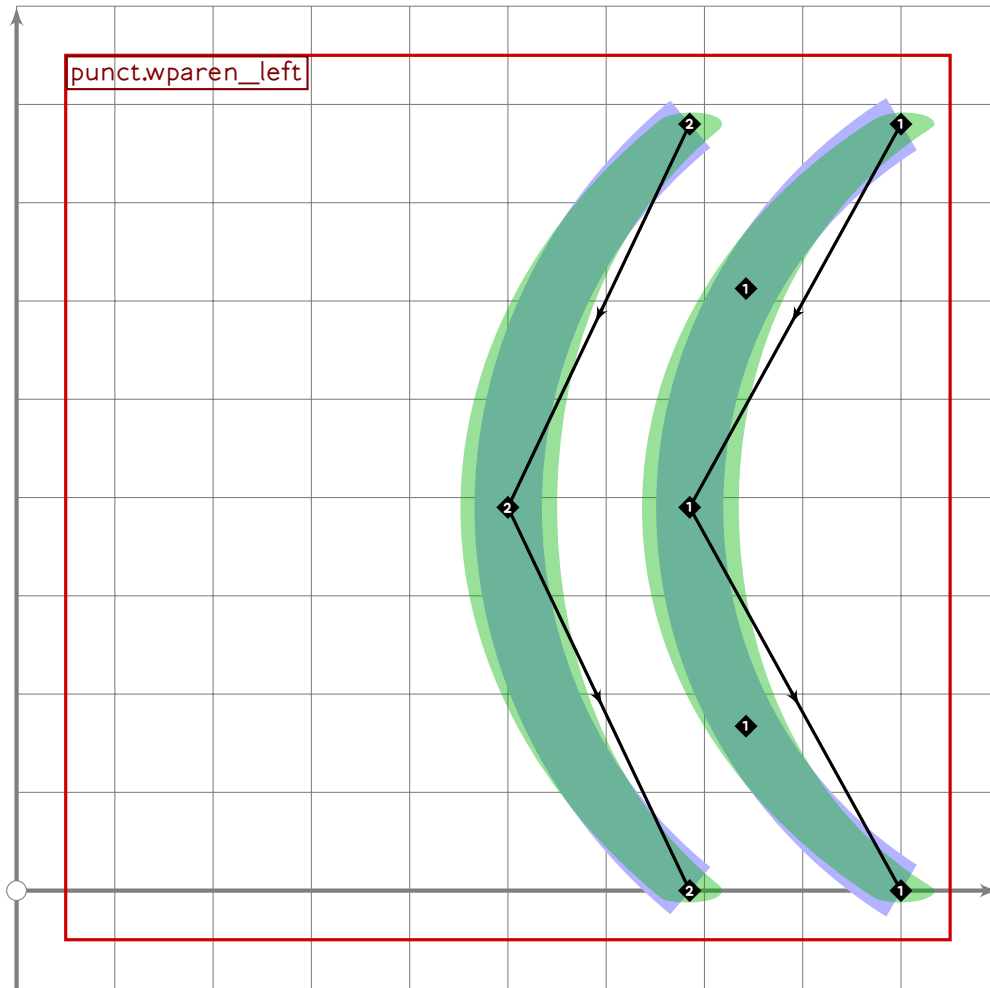
U+FF5E
tsuku.uniFF5E



```

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;

```

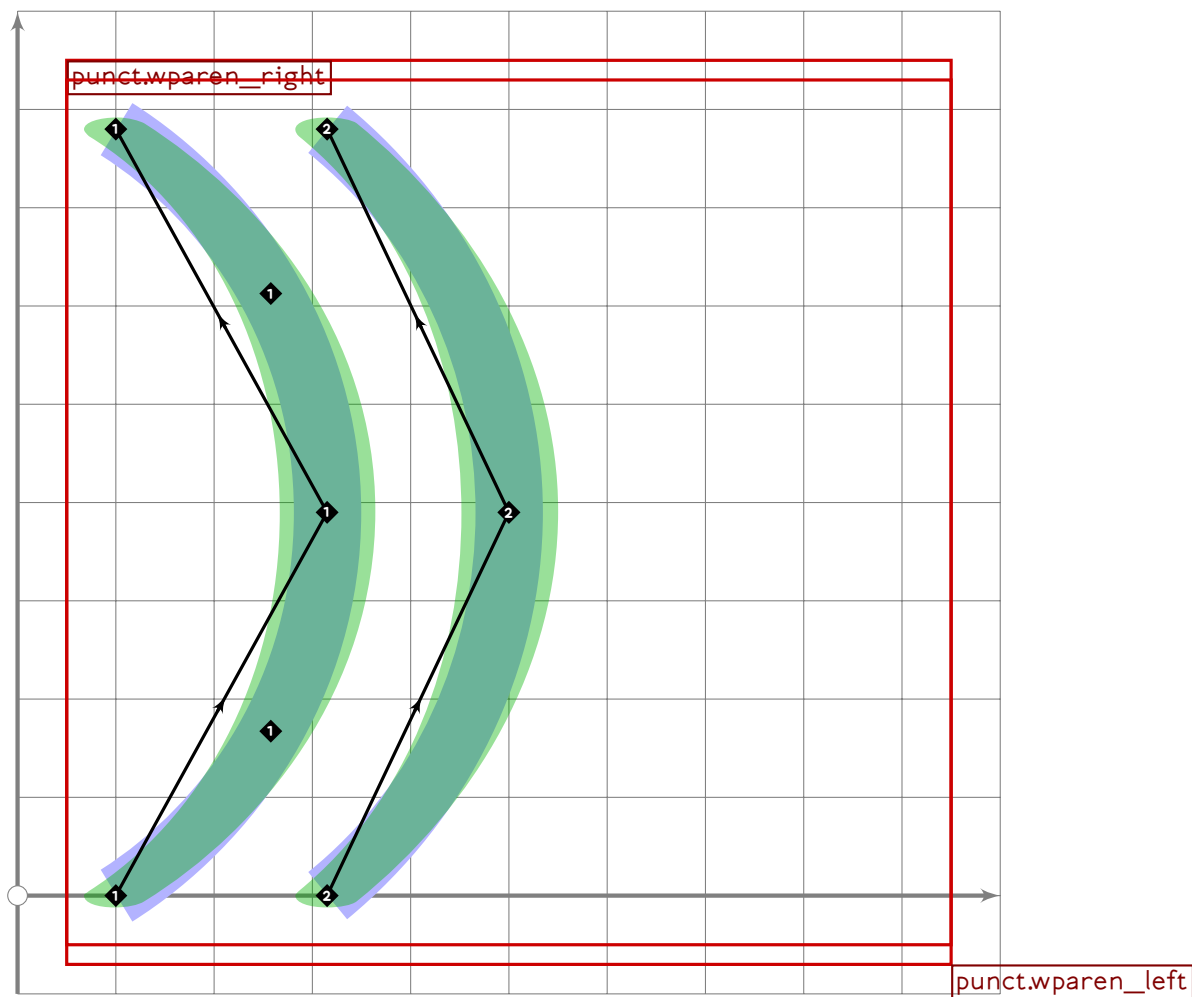


```

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;

```


U+FF60
tsuku.uniFF60



```

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 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 endif;
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 endif;
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[ngls]:=regenerate(serif shifted plp);
103     ngls:=ngls+1;
104   endgroup;
105 endif;
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[ngls]:=regenerate(serif shifted
117       (plp+(-1,0)*0.25*bts*(xpart dlp/abs(dlp))));
118     ngls:=ngls+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[ngls]:=regenerate(serif shifted (plp+0.25*bts*dlp/abs(dlp)));
132     ngls:=ngls+1;
133   endgroup;
134 enddef;
135
136 vardef tsu_serif.mincho.llpoint(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[ngls]:=regenerate(serif shifted plp);
146     ngls:=ngls+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[ngls]:=regenerate(serif shifted plp);
162     ngls:=ngls+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)*(yypart t)<0: serif:=reverse serif; fi;
191   glstk[ngls]:=regenerate(serif);
192   ngls:=ngls+1;
193 endgroup;
194 enddef;
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:=(subpath (0.2,2) of (z4{up}..z2..{down}z3))-cycle transformed t;
217   glstk[nxls]:=regenerate(serif);
218   nxls:=nxls+1;
219 endgroup;
220 enddef;
221
222 % standard serif codes:
223 % 1 - Latin left
224 % 2 - Latin right
225 % 3 - Latin left and right
226 % 4 - Mincho corner blob
227 % 5 - Mincho blob, pointy to ul
228 % 6 - Mincho blob, triangular
229 % 7 - Mincho blob, point to ll
230 % 8 - Mincho blob, point to l
231 % 9 - Mincho kanji triangle
232 % 10 - Mincho kanji half-ellipse
233
234 boolean tsu_do_serif[];
235
236 vardef tsu_serif.standard(expr bst,plp,dlp,l,bts,bos,b) =
237   if known tsu_do_serif[1]:
238     if tsu_do_serif[1] and (bst=1):
239       tsu_serif.latin.left(bst,plp,dlp,l,bts,bos,b);
240     fi;
241   fi;
242   if known tsu_do_serif[2]:
243     if tsu_do_serif[2] and (bst=2):
244       tsu_serif.latin.right(bst,plp,dlp,l,bts,bos,b);
245     fi;
246   fi;
247   if known tsu_do_serif[3]:
248     if tsu_do_serif[3] and (bst=3):
249       tsu_serif.latin.leftright(bst,plp,dlp,l,bts,bos,b);
250     fi;
251   fi;
252   if known tsu_do_serif[4]:
253     if tsu_do_serif[4] and (bst=4):
254       tsu_serif.mincho.corner(bst,plp,dlp,l,bts,bos,b);
255     fi;
256   fi;
257   if known tsu_do_serif[5]:
258     if tsu_do_serif[5] and (bst=5):
259       tsu_serif.mincho.ulpoint(bst,plp,dlp,l,bts,bos,b);
260     fi;
261   fi;
262   if known tsu_do_serif[6]:

```

```

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

```