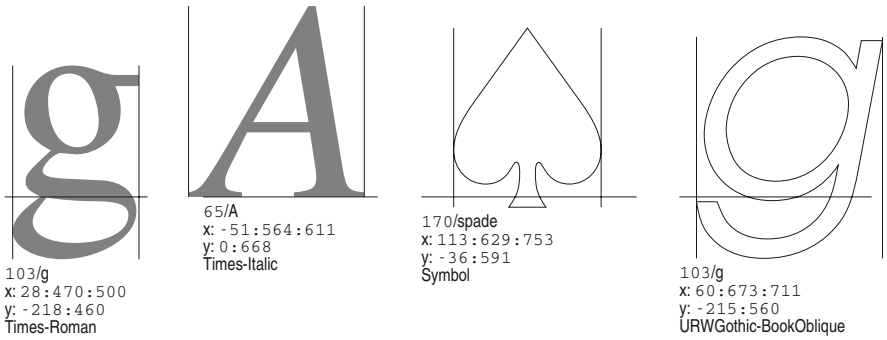
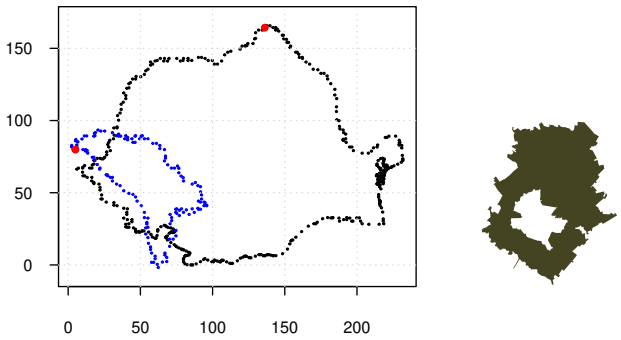
Caractere vechi și noi, cu PostScript, LaTeX și R

exerciții de programare

Vlad Bazon





iiaso 2020

Cuprins

vom demara *mai bine* (în orice caz – mai prudent): le înscriem deocamdată pe toate, într-un același director de lucru și vom încerca să producem pe seama lor (în directorul respectiv) un fișier LaTeX, urmând să vedem dacă `xelatex` (dar de această dată vom avea în vedere și `pdflatex`) îl compilează fără erori (caz în care vom putea muta „definitiv" fișierele, în locurile știute de TDS), sau încă nu (caz în care va trebui să vedem ce ar trebui adăugat sau modificat, în directorul de lucru curent, reluând apoi compilarea). Putem proceda așa, fiindcă directorul curent este favorizat, fiind de regulă *primul* consultat.

Constituim deci directorul în care vom lucra mai departe, `20-aug/` și copiem aici fișierul "ctrEU.pfb" (dar putem pleca de la orice alt fișier PFB, necomercial), stabilit în §9.1, redenumindu-l **`eureg.pfb`**:

```
mkdir 20-aug
cp ctrEU.pfb 20-aug/eureg.pfb
cd 20-aug    # de-acum încolo, vom lucra în acest director
```

Să creem și fișierul "`.raw`", prin care vom putea modifica fișierul PFB (înscriind la momentul potrivit, noul caracter):

```
t1disasm -o eureg.raw eureg.pfb
```

și chiar, imediat, să înlocuim în `eureg.raw` numele "`ctrEU`" cu "`eureg`" și să recreem fișierele PFB și PFA:

```
t1asm eureg.raw eureg.pfb
pfbtopfa eureg.pfb eureg.pfa
```

De-acum încolo, denumirea fontului (preluat din §9.1 cu numele "`ctrEU`") va fi "`eureg`".
În acest moment, în directorul `20-aug/` avem trei fișiere: **`eureg.pfa`** (pentru a folosi fontul `eureg` în programe PS – *vezi* §8.1), **`eureg.pfb`** (pentru a folosi fontul `eureg` în LaTeX) și **`eureg.raw`** (pentru a modifica, prin `t1asm`, fișierele PFB și PFA); pe parcurs vom adăuga fișierul metric **`eureg.afm`**, unele programe PS de testare și apoi, anumite fișiere LaTeX.

10.2 Un accent circumflex potrivit fontului

Având un accent circumflex în cadrul fontului, vom putea produce 'â' (pentru a scrie "România", cum doream mai sus), sau mai rău (ca aspect) 'â̲', sau de exemplu 'ĝ̲', etc.; nu-i nimic neobișnuit la aceste combinații de caractere: când scriem „de mână", întâi trasăm litera de bază și apoi adăugăm deasupra (sau dedesubt, după caz) accentul respectiv. Desigur că apar probleme de poziționare (*cât* de deasupra, la ce distanță de marginea stângă a literei suport) și dileme referitoare la forma accentului (nu-i prea „gros" sau prea „subțire", sau prea „uniform", față de contururile literelor existente?).

Prin FontForge ar fi poate ușor, de definit un accent circumflex (preluân-

du-l eventual dintr-un alt font); dar aici (în spiritul lucrării noastre) nu prea ne interesează „soluțiile" mecanice binecunoscute (fă dublu-click pe caracter, în fereastra apărută alege cu mouse-ul un „instrument" sau altul, fă click undeva, fă click-dreapta și folosește submeniul cutare, trage cu mouse-ul de linie, etc. etc.).

Ideea pe care o încercăm aici este următoarea: ne-am mări șansele de a obține un accent circumflex *compatibil* ca aspect cu literele existente, dacă îl derivăm cumva din jumătatea inferioară a conturului literei 'x'.
Mai întâi, prin următorul program PS conturăm șirul "ab" cu o mărime de font suficient de mare și apoi, reducând cumva mărimea fontului, conturăm un "x", undeva deasupra lui "a":

```
%!PS   20-aug/circf.ps
(eureg.pfa) run  % fontul eureg (din directorul de lucru curent)
    /ER {/eureg findfont exch scalefont} bind def
/draw { % <string> x y <sz> draw
    gsave
        ER setfont
        moveto  false charpath stroke  % true charpath fill
    grestore
} bind def
    72 72 translate
(ab) 0 0 360 draw  % cu mărimea de font 5×72
    -75 185 translate
(x) 82 0 288 draw  % cu mărimea de font 4×72
```

Desigur, am avut de experimentat puțin cu mărimile de font și cu parametrii de poziționare; am considerat și caracterul 'b' pentru a verifica dacă nu cumva accentul de deasupra lui 'a' (adică, deocamdată, jumătatea inferioară a lui 'x') se înalță prea sus față de marginea superioară a literelor 'b', 'h', etc. Considerând pentru o mai bună comparație, șirul "abehi" (în loc de "ab"), adăugând 'x' și deasupra lui 'e' și înjumătățind cele două mărimi de font în programul redat mai sus – rezultă (executând programul astfel modificat):

Am adăugat și 'i' fiindcă aceasta poate fi văzută și ca „literă cu accent" (întâi trasăm corpul, apoi adăugăm deasupra punctul), iar poziționarea existentă a punctului specific (cam la fel distanțat de corpul literei și de linia superioară

a literelor cu "ascender", 'b', 'd', 'h', 'f', 'k', 'l', 't') ne-ar putea servi ca model pentru a poziționa accentul circumflex pe care vrem să-l definim.

Acum avem nevoie de niște informații metrice (pentru a poziționa accentul *deasupra* literei) – deci să adăugăm în directorul de lucru fișierul AFM corespunzător fontului nostru:

```
pf2afm eureg.pfb
```

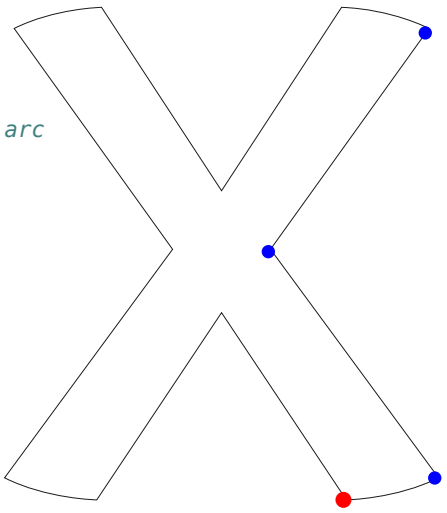
În fișierul rezultat `eureg.afm`, avem pentru 'a':

```
C 97 ; WX 500 ; N a ; B -20 -7 357 507 ;
```

deci „deasupra" lui 'a' înseamnă la o înălțime mai mare decât 507; putem observa mai general, că literele fără "ascender" (a, c, e, o, p, q, x, etc.) au cam aceeași înălțime deasupra liniei de bază (cea mai mică fiind 500, la 'z').

Avem nevoie de conturul literei 'x'; am putea folosi `pathforall` (am dat un mic exemplu în §3.2), pentru a obține secvența de instrucțiuni `lineto`, `curveto`, etc. care îl generează – dar ar trebui în final să ne ocupăm de transformări delicate asupra coordonatelor rezultate (dependente de mărimea de font folosită), pentru a obține coordonatele de caracter (întregi 0..1000); este mult mai simplu să preluăm direct definiția glifei `/x` (în sistemul de coordonate al caracterului), din fișierul `eureg.raw`:

```
/x {
    -42 504 hsbw
    15 -21 hstem
    507 -20 hstem
    356 -6 rmoveto  % punctul roşu
    0 0 52 1 44 22 rrcurveto  % primul arc
    -175 238 rlineto
    165 230 rlineto
    0 0 -38 20 -53 2 rrcurveto
    -125 -191 rlineto
    -125 191 rlineto
    0 0 -49 -1 -42 -21 rrcurveto
    165 -230 rlineto
    -175 -238 rlineto
    44 -22 52 -1 0 0 rrcurveto
    130 195 rlineto
    closepath
    endchar
} ND
```



(x) false charpath **stroke**

Procedura `/x` redată mai sus descrie conturul literei 'x', prin instrucțiuni specifice fonturilor "Type 1" (v. [AdT1]); se pleacă din punctul (356, -6) (marcat cu roșu), se trasează un arc de curbă până în punctul marcat cu albastru (prin prima instrucțiune `rrcurveto`), apoi se continuă cu cele două segmen-

te din dreapta literei, urmează al doilea arc de curbă (prin a doua instrucțiune `rrcurveto`), continuând apoi în sens *antiorar*, până ce `closepath` încheie traseul (revenind în punctul roșu).

Din acest contur ne interesează aici numai primul și al patrulea arc, împreună cu cele patru segmente conectate acestora (formând jumătatea inferioară a conturului); vom formula acest subcontur pe o cale mai ocolită (ar fi nevoie de ceva dexteritate, pentru a-l formula direct), încheiată abia prin §10.4.

Să obținem întâi—după [AdT1], §6.4 *Charstring Command List*—instrucțiunile PS **`rcurveto`** corespunzătoare instrucțiunilor **`rrcurveto`** din definiția `/x` redată mai sus:

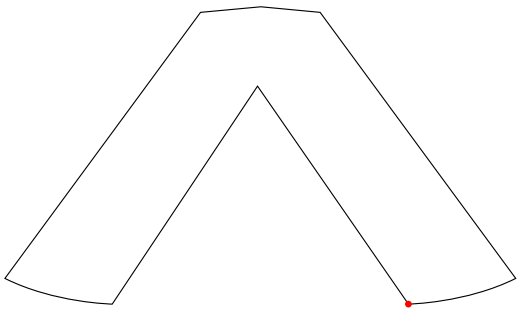
dx1	dy1	dx2	dy2	dx3	dy3	rrcurveto	(șablonul comenzii)
0	0	52	1	44	22	rrcurveto	(primul arc)
dx1	dy1	dx1+2	dy1+2	dx1+2+3	dy1+2+3	rcurveto	(în PS)
0	0	52	1	96 (52+44)	23 (1+22)	rcurveto	(primul arc, în PS)

Aici, "dx" și "dy" desemnează deplasări pe orizontală și pe verticală, față de punctul curent (iar "dx1+2" de exemplu, însumează "dx1" și "dx2").

Deci primul arc se trasează în PS prin: `0 0 52 1 96 23 rcurveto`; analog, găsim pentru al patrulea arc: `44 -22 96 -23 96 -23 rcurveto`.

Putem „traduce" în PS întreaga procedură `/x` redată mai sus (ignorând `hsbw`, `hstem` și `endchar`) și intercalând "`currentpoint == == `" putem afla coordonatele celor două puncte de la mijlocul conturului; distanța dintre ele este de 107. Reținând apoi numai jumătatea inferioară a conturului și unind punctele din mijloc printr-o linie ușor frântă la mijlocul ei, obținem:

```
%!PS
/semix {
    356 514 moveto
    0 0 52 1 96 23 rcurveto
    -175 238 rlineto
    -53 5 rlineto
    -54 -5 rlineto
    -175 -238 rlineto
    44 -22 96 -23 96 -23 rcurveto
    130 195 rlineto
    closepath
} bind def
    72 0 translate
semix stroke
```



un „accent circumflex"

De observat că am ridicat accentul la înălțimea 520 (cum am văzut mai sus, literele mici fără "ascender" urcă până la înălțimea 507) – încât instrucțiunea inițială `356 -6 rmoveto` a devenit acum `356 514 moveto`.

Reducerea conturului

Ca mărime, accentul rezultat este „jumătate din x" – ceea ce este totuși prea mult (dacă 'x' și litera de bază sunt redate cu o *aceeași* mărime de font); mai sus (în "circf.ps"), când am așezat 'x' deasupra literelor 'a' și 'e', am folosit mărimi de font diferite, **5**×72 pentru literele de bază și **4**×72 pentru 'x'.

Cum am reduce acum, cu factorul $4/5 = 0.8$, conturul rezultat mai sus prin procedura `/semix`? Păi n-avem decât să multiplicăm cu factorul 0.8, toate valorile din instrucțiunile `rlineto` și `rcurveto`, din corpul procedurii `/semix`; pentru aceasta, copiem într-un tablou PS valorile respective, în ordinea *inversă* a liniilor pe care apar (începând de la linia dinaintea lui `closepath`) și folosim `forall` pentru a modifica elementele tabloului:

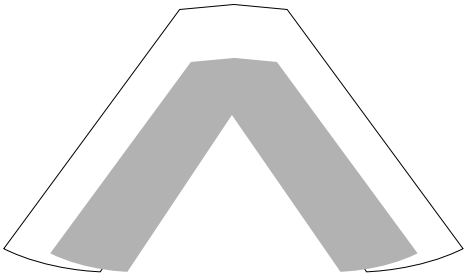
```
      gs -q -dNODISPLAY
GS> [130 195 44 -22 96 -23 96 -23 -175 -238 -54 -5 -53 5 -175 238
0 0 52 1 96 23]
      GS<1> {.8 mul} forall 22 array astore ==
[104.0 156.0 35.2 -17.6 76.8 -18.4 76.8 -18.4 -140.0 -190.400009 -43.2
-4.0 -42.4 4.0 -140.0 190.400009 0.0 0.0 41.6000023 0.8 76.8 18.4]
```

Presupunând că avem pe stiva operanzilor elementele tabloului inițial, operatorii din secvența *`rcurveto rlineto rlineto ...`* vor consuma pe rând valorile respective (începând din vârful tabloului), conducând la executarea instrucțiunilor `0 0 52 1 96 23` *`rcurveto`*, apoi `-175 238` *`rlineto`* ș.a.m.d. din programul `/semix` – încât acesta poate fi rescris în forma redată mai jos; procedăm analog pentru tabloul obținut mai sus, al valorilor multiplicate cu 0.8 – formulând procedura `/semix1`:

```
%!PS    smx.ps
/semix {  % conturul inițial
    356 514 moveto
    130 195  44 -22 96 -23 96 -23  -175 -238  -54 -5  -53 5
    -175 238  0 0 52 1 96 23
    rcurveto rlineto rlineto rlineto rlineto rcurveto rlineto
    closepath  stroke
} bind def
/semix1 {  % reduce conturul cu factorul 0.8
    330 514 moveto
    104.0 156.0 35.2 -17.6 76.8 -18.4 76.8 -18.4 -140.0 -190.400009
    -43.2 -4.0 -42.4 4.0 -140.0 190.400009 0.0 0.0 41.6000023
    0.8 76.8 18.4
    rcurveto rlineto rlineto rlineto rlineto rcurveto rlineto
    closepath  fill
} bind def
```

```
72 36 translate
semix

.7 setgray
semix1
```



Am evitat să multiplicăm cu 0.8 și valorile din instrucțiunea `moveto`, cu care începe trasarea conturului – vrând să păstrăm aceeași ordonată inițială, 514; însă am deplasat puțin spre stânga abscisa inițială (mutând-o, prin încercări, de la 356 la 330), încât conturul redus dat de `/semix1` (cel „plin" din imagine) să fie centrat aici, față de cel produs inițial prin `/semix`.

10.3 Extinderea dicționarului *CharStrings* al fontului

Ghostscript ne permite (spre deosebire, aflăm, de alte interpretoare de PS) să adăugăm un nou caracter, în dicționarul fontului (dar nu pentru orice font); mai mult, noua glifă poate fi definită chiar în PS – nu este necesar să transformăm instrucțiunile PS `rcurveto` în instrucțiuni Type-1 `rrcurveto`, încât vom putea adăuga chiar definiția `/semix1` de mai sus.

Constituim „pas cu pas" programul *addChar.ps*, apelând din când în când la câte o sesiune de lucru interactiv cu GS, pentru a ne lămuri și preciza anumite aspecte:

```
%!PS    addChar.ps
(eureg.pfa) run
```

Prin încărcarea fișierului-font "eureg.pfa" se creează o intrare în dicționarul `/FontDirectory`, prin care se asociază numele de font indicat în fișier, cu dicționarul de font rezultat prin execuția fișierului-font respectiv:

```
gs -q -dNODISPLAY addChar.ps
GS> FontDirectory {exch == ==} forall
/eureg   % numele fontului încărcat
-dict-   % referință la dicționarul fontului
```

Acum, `findfont` găsește în `FontDirectory` cheia `/eureg` și încarcă pe stiva operanzilor, dicționarul asociat ei:

```
/eureg findfont  % pe stivă: <D1> := dicţionarul fontului
```

Dicționarul fontului—tocmai adus în stivă, notându-l pentru simplitate cu "D1"—nu poate fi modificat direct – ceea ce putem verifica prin operatorul **`wcheck`** (returnează "false", deci accesul pe D1 este "*read-only*"); aflăm numărul de chei ale lui D1 (`dup length ==` ne dă 13) și prin **`dict`** constituim în vârful stivei un dicționar "D2" cu același număr de chei, în care apoi, copiem elementele din D1 (după copiere, D1 va fi eliminat din stivă):

```
dup length dict  copy  % pe stivă: <D2> := copie inscriptibilă a lui D1
```

Putem constata că pe D2 avem „dreptul de scriere”: `dup wcheck ==` ne dă ”true”; dar acest drept este oarecum limitat. Putem înscrie direct o nouă valoare pentru o cheie „simplă”, precum `/PaintType` (v. §8.3):

```
gs -q -dNODISPLAY addChar.ps
GS<1> dup /PaintType 2 put  % înscrie 2 în loc de valoarea inițială 0
GS<1> /PaintType get ==     % din dicționarul D2
2   % noua valoare, în D2
GS> /eureg findfont /PaintType get ==
0   % în dicționarul fontului, valoarea a rămas 0
```

Însă dacă valoarea inițială este un tablou PS, sau un dicționar (și nu un simplu număr, ca pentru `/PaintType`), atunci „valoarea” respectivă este de fapt o *referință* la valoarea originală (existentă în dicționarul din care s-a copiat) și este iarăși ”read-only”:

```
gs -q -dNODISPLAY addChar.ps
GS<1> /Encoding get wcheck ==
false  % read-only
```

Prin urmare, dacă vrem să înscriem în `/Encoding`, sau în `CharStrings`, atunci va trebui să procedăm la fel ca și cu dicționarul fontului (creând câte o copie inscriptibilă, analog cu ”D2” de mai sus).

Continuăm ”addChar.ps” cu:

```
begin
```

prin care D2 (copia inscriptibilă a dicționarului fontului) va fi mutat din stivă, în vârful stivei dicționarelor, devenind „dicționarul *curent*”; după aceasta, vom putea accesa direct o valoare sau alta din acest dicționar, prin numele cheii (regula este de a căuta întâi în dicționarul *curent* și numai când cheia indicată nu este găsită aici, ea va fi căutată în celelalte dicționare din stiva dicționarelor).

Prin următoarele două definiții, accesăm din dicționarul curent tabloul de sub cheia `/Encoding` și respectiv, dicționarul de sub cheia `/CharStrings`, creem în stivă câte o copie inscriptibilă și în final (prin efectul operatorului **`def`**) înscriem aceste copii în dicționarul *curent* (înlocuind practic, valorile ”read-only” existente inițial sub cheile respective):

```
/Encoding  Encoding dup length array copy  def
/CharStrings  CharStrings dup length dict copy  def
```

Putem verifica imediat că vom putea înscrie:

```
gs -q -dNODISPLAY addChar.ps
GS> Encoding wcheck ==
true  % avem drept de scriere, în noul tablou
```

Folosindu-ne de **put**, înscriem cheia /semix pe locul 1 (sau pe un alt loc, ocupat inițial de /.notdef) din tabloul Encoding și introducem în dicționarul CharStrings cheia /semix, asociindu-i ca valoare procedura PS care ne-a produs mai sus "accent circumflex" (cu instrucțiunile din /semix1):

```
Encoding  1 /semix  put
CharStrings  /semix {
    330 514 moveto
    104.0 156.0 35.2 -17.6 76.8 -18.4 76.8 -18.4 -140.0 -190.400009
    -43.2 -4.0 -42.4 4.0 -140.0 190.400009 0.0 0.0 41.6000023
    0.8 76.8 18.4
    rcurveto rlineto rlineto rlineto rlineto rcurveto rlineto
    closepath  fill
} bind  put
```

Transformăm acum dicționarul curent, astfel modificat, în dicționar de font (prin **definefont**), înscriindu-l în stiva dicționarelor de font sub un nou nume (să zicem, /Neureg); curățăm apoi stiva (definefont verificase corectitudinea și lăsase pe stivă dicționarul de font creat) și în final, eliminăm din stiva dicționarelor, dicționarul curent (prin **end**):

```
/Neureg currentdict definefont pop
end
```

Acum putem încheia fișierul *addChar.ps*, punându-l la dispoziția oricărui program PS care angajează fontul /eureg și are nevoie de un caracter suplimentar, "accent circumflex" – ca în acest mic experiment:

```
%!PS    addCharTest.ps
(addChar.ps) run
    /SZ 36 def
    /regl {SZ 1000 div mul} bind def
    /Neureg findfont //SZ scalefont setfont
72 600 moveto  .5 setgray
    (Roma) show
gsave  % reglează poziția accentului
    542 regl neg 0  rmoveto
    (\001) show  % scrie accentul
grestore
    (nia) show
3 3 scale  .6 setgray
    /ROU glyphshow
.1 setgray .1 setlinewidth
    (\242) false charpath stroke
```

Știm din fișierul metric **eureg.afm** (v. §10.2) că 'a' are lățimea 500 (în sistemul de coordonate al caracterului); pentru a plasa accentul deasupra, am

mutat punctul curent puțin la stânga originii lui 'a', cu 542×SZ/1000 (transformând de la sistemul de coordonate al caracterului la sistemul uzual, ținând seama de mărimea /SZ a fontului).

Desigur că am probat și cu alte mărimi /SZ (și cu alte litere – e, o, u); rezultatul grafic pare satisfăcător. Dar maniera care s-ar impune pentru folosirea ca accent a noului caracter este neconvenabilă, trebuind să-i „reglăm" poziția (și anume, într-un sub-context grafic temporar, pentru a păstra „punctul curent" la care show are de scris literele obișnuite).

Pe de altă parte, lansând ca de obicei programul redat mai sus, prin **gs** -q addCharTest.ps – avem o surpriză cam inexplicabilă, în finalul execuției: pe stivă rămâne o valoare, anume 1; dar era cumva de așteptat să dăm peste vreo situație de *excepție*, dat fiind că accentul nostru (de cod 1, ca și valoarea rămasă în stivă) a fost introdus (spre deosebire de caracterele existente) printr-o procedură PS obișnuită (*nu* "Type-1") și încă, fără vreo indicație de *lățime* (pe baza căreia show să poată stabili unde să traseze caracterul următor).

10.4 Adăugarea noii glife în fișierul PFB

Modificând fișierul **eureg.raw** și folosind programul t1asm, putem înscrie accentul nostru (după anumite ajustări ale procedurii PS prin care l-am definit mai sus) direct în fișierul-font **eureg.pfb** (cu avantajul, față de §10.3, că astfel vom putea folosi fontul eureg și în LaTeX de exemplu, nu numai în programe PS).

În fișierele PFB, contururile caracterelor trebuie definite prin instrucțiuni specifice formatului Type-1 (în particular, folosind rrcurveto și nu rcurveto); în plus, coordonatele trebuie să fie numere întregi (dar valorile care nu sunt întregi pot fi „ajustate" folosind operatorul div; de exemplu 17.6 poate fi introdus prin 176 10 **div**).

În procedura /semix din programul "addChar.ps" redat mai sus, avem două instrucțiuni rcurveto; să le transformăm în rrcurveto:

```
0.0          dx1                          0
0.0          dy1                          0
41.6000023   dx1+2       dx2 ≈ 41.6-0.0   416 10 div
0.8          dy1+2       dy2 = 0.8-0.0    8 10 div
76.8         (dx1+2)+3   dx3 ≈ 76.8-41.6  352 10 div
18.4         (dy1+2)+3   dy3 = 18.4-0.8   176 10 div
rcurveto                                  rrcurveto
```

Analog pentru a doua, 35.2 -17.6 76.8 -18.4 76.8 -18.4 **rcurveto**, găsim 352 10 div -176 10 div 416 10 div -8 10 div 0 0 **rrcurveto**.

Deschidem în gedit fișierul eureg.raw, căutăm cheia /Encoding și înscriem la indexul 1 cheia /semix:

```
% /Encoding StandardEncoding def
/Encoding 256 array 0 1 255 {1 index exch /.notdef put} for
dup 1 /semix put
dup 32 /space put
dup 65 /A put
%    ...
```

Căutăm cheia `CharStrings` și mai întâi, înlocuim 93 cu 94 – reflectând faptul că vom adăuga un nou caracter:

```
2 index /CharStrings 94 dict dup begin
```

Obs. De observat că în *addChar.ps* din §10.3, am prevăzut `/CharStrings` de *aceeași* lungime cu dicționarul original, deși urma să-i adăugăm încă o intrare – ceea ce era totuși corect, fiindcă în PS-*Level 2*, dicționarele sunt extinse în mod automat, când se adaugă o nouă intrare. Însă formatul "Type-1" recunoaște numai o anumită parte a limbajului PostScript, așa că măcar din prudență, trebuie să declarăm lungimea exactă a conținutului etichetat în fișier de `/CharStrings`.

La sfârșitul fișierului `eureg.raw`, imediat deasupra liniei *`end end`*, adăugăm definiția caracterului nou `/semix` (implicând instrucțiunile `rrcurveto` stabilite mai sus și adaptând prin `10 div` coordonatele neîntregi din instrucțiunile `rlineto` preluate de la procedura `/semix` din §10.3):

```
/semix {
    0 500 hsbw  % "horizontal sidebearing"=0, "width"=500
    280 514 rmoveto  % punctul de start al conturului
    0 0 416 10 div 8 10 div 352 10 div 176 10 div  rrcurveto
    -140 1904 10 div  rlineto
    -424 10 div 4  rlineto
    -432 10 div -4 rlineto
    -140 -1904 10 div rlineto
    352 10 div -176 10 div 416 10 div -8 10 div 0 0  rrcurveto
    104 156  rlineto
    closepath
    endchar  % marchează încheierea definiţiei glifei
} ND
```

De observat că am modificat abscisa punctului de start al conturului, față de procedurile `/semix` din §10.3 (unde o fixasem 356 sau 330 și trebuia apoi să „reglăm" poziția accentului – v. "addCharTest.ps"); acum (conturând din punctul (280, 514)), accentul se va întinde spre stânga până foarte aproape de marginea stângă a caracterului deasupra căruia ar fi scris, încât nu mai este necesar niciun reglaj.
De observat deasemenea, că n-am mai adăugat `fill` (după `closepath`) – pentru că modul de redare „plin" sau „contur" este stabilit la începutul fișieru-

lui, pentru toate caracterele, prin variabila `/PaintType`.

După cele trei modificări prezentate mai sus, salvăm fișierul `eureg.raw` și producem prin `t1asm` și `pfbtopfa`, fișierele PFB și PFA:

```
t1asm eureg.raw eureg.pfb
pfbtopfa eureg.pfb eureg.pfa
```

Putem verifica prin FontForge fișierele-font rezultate, după cum putem cataloga glifele respective și prin diversele programe menționate deja sau constituite aici, anterior. Să testăm folosirea fontului într-un program PS:

```
%!PS
(eureg.pfa) run
/eureg findfont 48 scalefont setfont
.4 0 0 72 72 translate moveto setgray
(Rom) show
gsave (a) show grestore
(\001nia\242) show
```

Am pus `(a) show` într-un sub-context *gsave*/*grestore* pentru a conserva „punctul curent”: 'a' și accentul `\001` care îi urmează, vor avea cumva aceeași origine (în privința abscisei; ordonata originii accentului este puțin deasupra conturului lui 'a' – cum și trebuie să fie).

Putem constata, introducând:

```
/circf {  % pe stivă: caracterul de accentuat
    gsave show grestore  % scrie şi conservă originea
    (\001 ) show  % adaugă accentul şi mută originea cu un spaţiu
} bind def
    72 144 moveto
[(x) (e) (o) (p) (s) (z) (g) (n) (y)] {circf} forall
```

că accentul nostru „stă bine” peste litere ca a, o, e, z, n, p; stă mai puțin bine peste y sau x – ceea ce s-ar putea îndrepta urcând puțin accentul (la 520 de exemplu, în loc de 514); stă foarte rău (fără ajustări suplimentare) peste litere largi, m sau w, ca și pe literele înguste, i sau j.

Dacă în loc de `/semix` am fi folosit numele standard `/circumflex` și dacă l-am fi plasat în `/Encoding` nu la indexul 1, ci la indexul 94 (ca în codificarea standard a caracterelor) – atunci puteam folosi comanda **`seac`** (v. [AdT1]) pentru a adăuga în `eureg.raw`, la sfârșitul listei "CharStrings", definiții pentru literele accentuate, indicând poziția accentului față de originea caracterului. De exemplu, **`/ycircumflex`**`{0 280 520 121 94` **`seac`**`}` definește un *nou* caracter, „y cu accent circumflex” (121 și 94 fiind codurile ASCII pentru 'y' și accent); ordonata originii accentului ar fi acum 520 (în loc de 514).

tual, adăugăm ceva *opțiuni* în `rojud.sty`), putem încerca să depunem arhiva asociată, la CTAN (acceptând și riscul de a fi respins).

Pentru „testare" ar fi ușor de adaptat programele din §9.3, în care foloseam `\ifcase` și `\minipage`, precum și comenzi din pachetele *`pgffor`* și *`ifthen`* – întrebându-ne totuși dacă n-ar fi mai simplu de tabelat *direct* glifele respective.

Preferăm acum să ne bazăm numai pe mediul LaTeX *`tabular`*. Alegem să tabelăm glifele câte 7 pe un rând, iar dedesubtul fiecărui rând de glife adăugăm un rând conținând numele județelor reprezentate de glifele respective; dar bineînțeles că nu vom formula manual conținutul tabelului respectiv (ar fi cum nu se poate mai incomod), ci îl vom genera într-un fișier separat *`rows.tex`*, printr-un mic program Python (sau **R**, etc.).

Programul `test_rojud.tex` este chiar foarte simplu:

```
\documentclass[12pt]{article}
    \usepackage{rojud}  % pachetul de testat (fontul judeţelor)
    \usepackage{xcolor}  % dacă vrem să colorăm glifele
    \newcommand{\setFont}[1]{\fontfamily{#1}\selectfont}
\begin{document}
    \begin{tabular}{*{7}{c}}  % 7 coloane, conţinut centrat
        \input{rows.tex}  % inserează rândurile tabelului
    \end{tabular}
\end{document}
```

Prin comanda `\setFont` vom putea putea seta ca „font curent" fie `rojud`, pentru rândurile de glife ale județelor, fie un alt font (conținând litere) pentru rândurile de nume ale județelor; dar în formularea fișierului `rows.tex` va trebui să ținem seama de acest aspect: în *`tabular`* celulele tabelului formează grupuri individuale, independente între ele în privința atributelor stilistice – altfel spus, nu putem seta fontul pentru întregul rând, ci trebuie să-l specificăm pe fiecare celulă a rândului.

Pentru a genera fișierul `rows.tex`, preluăm lista *`std`* din programul anterior `dup.py` (dar eliminăm prefixul '`j`' din numele județelor), o împărțim în 6 subliste de câte 7 nume și apoi, pentru fiecare sublistă constituim cele două rânduri necesare în mediul LaTeX `tabular`. Celulele trebuie despărțite prin '`&`' (desigur, vom elimina '`&`' din fața *primei* celule a rândului). Pentru o glifă (de exemplu pentru județul Alba), celula are forma

```
\setFont{rojud}\Huge\color[HTML]{8080FF}{\jAL}
```

iar pentru numele corespunzător:

```
\setFont{lmtt}\footnotesize{AL}
```

unde '`lmtt`' este numele LaTeX al fontului "*Latin Modern Typewriter*".

Programul Python care generează rândurile în modul descris mai sus

(afișându-le pe ecran) se poate formula astfel:

```
std = ["AB","AR","AG","BC","BH","BN","BT","BV","BR","BI","BZ",
       "CL","CS","CJ","CT","CV","DB","DJ","GL","GR","GJ","HR","HD",
       "IS","IL","IF","MM","MH","MS","NT","OT","PH","SJ","SM","SB",
       "SV","TR","TM","TL","VL","VS","VN"]
L6Lj = [std[i:i+7] for i in range(0, 42, 7)]  # print(L6Lj)
for Lj in L6Lj:
    row1 = [' & \\setFont{rojud}\\Huge\\color[HTML]{8080FF}{\\j' +
             jud + '}' for jud in Lj]
    row1[0] = row1[0][3:]  # elimină '&' de la începutul rândului
    row2 = [' & \\setFont{lmtt}\\footnotesize{' + jud + '}'
            for jud in Lj]
    row2[0] = row2[0][3:]
    print(''.join(row1) + ' \\\\')  # rândul glifelor
    print(''.join(row2) + ' \\\\')  # rândul numelor de judeţ
```

După ce verificăm pe ecran că n-am greșit nimic, relansăm programul, redirectând ieșirea pe fișierul anunțat deja, *rows.tex*; după aceasta, programul *test_rojud.tex* (în care foloseam prin **\input** fișierul rows.tex) poate fi compilat, fie prin xelatex, fie prin pdflatex și rezultă: