Fontologia

➢

Fontowe ABC

Bogusław Jackowski
i Stanisław Wawrykiewicz

Co jakiś czas na liście dyskusyjnej GUST-u padają głosy świadczące z jednej strony o dużym zainteresowaniu sprawami fontowymi, a z drugiej strony – o częstym niezrozumieniu podstawowych spraw związanych z fontami.

Zainteresowanie fontami jest o tyle uzasadnione, że ostatnimi czasy pojawia się na rynku coraz więcej fontów PostScript-owych, dających się wykorzystać do składania polskich tekstów – ot, choćby fonty dystrybuowane przez firmę ThETA.

Te aspekty skłoniły nas do podjęcia próby spisania fontowego elementarza dla tych dzielnych adeptów fontologii, którzy nie boją się pułapek, od których niestety aż się roi w świecie fontów.

Fonty „TEX-owe” czyli pliki TFM. TEX-owi jest absolutnie obojętne, jakich fontów używamy: czy to są mapy bitowe, fonty PostScript-owe w formacie Type 1, czy też może egzotyczne fonty „własnego chowu” – nie ma to znaczenia. Jedyne, czego potrzebuje TEX, to informacja metryczna, czyli pliki TFM (TEX Font Metric). Pliki metryczne, jak sama nazwa wskazuje, zawierają w pierwszym rzędzie informację o rozmiarach znaków, ale także informację o wielkości kernów (podcięć) dla par znaków, informację o wielkości korekty kursywy (*italic correction*) dla poszczególnych znaków oraz – choć trudno to zakwalifikować jako informację metryczną – informację o ligaturach, tzn. informację, które pary znaków mają być zastępowane automatycznie innym znakiem, np. dwa umieszczone obok siebie znaki -- zamienione zostaną przez TEX-a przez pojedynczy znak „–” zwany pauzą półfiretową (*endash*).

Krótko mówiąc, żeby w ogóle przetworzyć dokument TEX-em, potrzebne są pliki TFM.

Fonty „właściwe”. Dopiero sterowniki zamieniając złożone dokumenty, czyli pliki DVI, na postać widoczną już to na ekranie, już to na papierze, już to na jakimś innym nośniku, potrzebują tych „właściwych fontów”.

Fonty rodziny Computer Modern. Zgodnie z konwencją przyjętą przez Donalda E. Knutha nazwy fontów rodziny Computer Modern zawierają formalny rozmiar fontu w punktach, np. cmr10 (font 10-punktowy), cmbx7 (font 7-punktowy), itp. Należy tu podkreślić, że font cmr12 nie jest równoważny fontowi zdefiniowanemu jako cmr10 at 12pt – chociaż oba fonty mają wielkość 12 punktów, to jednak proporcje znaków w obu fontach są inne. Tę samą cechę dziedziczą oczywiście fonty wywodzące się z rodziny Computer Modern (Computer Concrete, EC, PL), a także fonty rodziny Euler, fonty cyryliczne, i inne fonty tworzone w duchu konwencji rodziny Computer Modern.

Zmiany proporcji wraz ze zmianą stopnia pisma są jak najbardziej uzasadnione, małe fonty powinny dla lepszej czytelności mieć nieco większe i szersze oczka minuskuł w stosunku do majuskuł. Niemniej jednak wydaje się, że Donald E. Knuth nieco przesadził z liczbą odmian fontów zmieniając co jeden punkt proporcje znaków.

W praktyce wystarczyłyby nie więcej niż cztery zakresy: pierwszy dla fontów mniejszych niż 8-punktowe, drugi dla fontów od 8 do 12 punktów, trzeci dla fontów od 12 do – powiedzmy – 24 punktów i wreszcie czwarty dla fontów tzw. nagłówkowych. Ostatni z wymienionych zakresów nie pojawia się w fontach Computer Modern. Font calowy cminch, przeznaczony do nagłówków, jest w istocie przeskalowanym proporcjonalnie fontem cmssbx10. Innymi słowy dokładnie ten sam efekt uzyskałoby się za pomocą instrukcji \font \f=cminch i za pomocą instrukcji \font \f=cmssbx10 at 1.44in (sic!). Jedyna różnica polega na tym, że font cminch jest nieco „oszczędniejszy” – zawiera tylko duże litery i cyfry.

Sporo zamieszania wywołały także plain-owe instrukcje \magstephalf i \magstep1, \magstep2, . . . , \magstep5, sugerujące, że jedyne „legalne” powiększenia fontów to 109,5%, 120%, 144%, 172,8%, 207,4% lub co najwyżej 248,8% (p. *The TEXbook*, str. 17). Mimo iż nie jest to nigdzie *expressis verbis* powiedziane, to jednak sporo nieporozumień narosło na tym tle. Trudno się temu dziwić, skoro prawie wszystkie pakiety zawierające fonty TEX-owe w postaci map bitowych (p. niżej) stosowały się do tej reguły, tzn. mapy bitowe generowane były tylko dla wymienionych wyżej powiększeń. Chcąc w dokumentach TEX-owych w sposób jednolity korzystać z fontów tradycyjnych, czyli bitmapowych, i z fontów PostScript-owych, nie należy przejmować się zanadto rozpowszechnionym obyczajem mającym swe korzenie w czasach, gdy komputery były wolniejsze i wygenerowanie fontu „od ręki" było zbyt kosztowne. Rzadko używana instrukcja postaci \font ... at ... jest w tym kontekście godna polecenia.

Fonty bitmapowe. W tradycyjnej konfiguracji systemu TEX podstawowymi fontami są mapy bitowe (pliki GF – *generic font*) generowane przez program METAFONT. Najczęściej używanym formatem jest jednak nie format GF, a nieco oszczędniejszy format PK (*packed*). Do zamiany służy program GFTOPK, z reguły wywoływany automatycznie w procesie generowania fontów.

Fonty bitmapowe generowane są dla *konkretnej wielkości pisma i konkretnego urządzenia*, tzn. zawierają pewne informacje pozwalające na danym urządzeniu uzyskać optymalny wygląd znaków. Stąd np. odrębne fonty dla drukarek laserowych i drukarek atramentowych o tej samej rozdzielczości 300 dpi.

TEX – jak już wspomnieliśmy – pozwala skalować całość dokumentu lub poszczególne fonty. Na przykład to samo przeskalowanie fontu można wyrazić na (co najmniej) trzy różne sposoby:

```
\font\A plr10 scaled 1728
\font\B plr10 scaled \magstep3
\font\C plr10 at 17,28pt
```

gdzie plr10 oznacza nazwę fizycznie posiadanego na dysku pliku plr10.tfm. TEX przeskaluje font plr10 tak, jak sobie tego życzymy, a ściślej – przeskaluje informację metryczną związaną z fontem. Żeby móc zobaczyć, jak wyglądają znaki przeskalowanego fontu, należy skorzystać z METAFONT-a, który oczywiście potrafi generować fonty dowolnie przeskalowane.

Fonty PK zwykle przechowywane są w katalogach o nazwach zawierających liczbę wynikającą z przemnożenia rozdzielczości przez przeskalowanie, np. fonty o rozdzielczości 300 dpi (*dots per inch*) powiększone o 20% (\magstep1) znajdą się w katalogu o nazwie 360\, 360dpi\, dpi360 lub jakimś innym, podobnie nazwanym. Jeśli system operacyjny zezwala na nadawanie plikom długich nazw, to przeskalowanie może być wpisane w nazwę pliku, np. fonts/pk/laserjet/pl/plr10.360pk i zbędne są wtedy odrębne podkatalogi.

Standardowe sterowniki przetwarzające pliki DVI czy to na postać wyświetlaną na ekranie, czy też drukowaną, używają fontów PK i posiadają mechanizm przeszukiwania odpowiedniej struktury katalogów lub interpretacji nazw fontów przeskalowanych. Sterowniki te pozwalają zwykle na bardzo szybkie wyświetlenie składanego tekstu. Współczesne sterowniki posiadają także możliwość współpracy z METAFONT-em – potrafią przekazać informację o brakujących fontach i uruchomić METAFONT-a, wstrzymując pracę do czasu zakończenia procesu generowania brakujących fontów. Dzięki temu nie ma obecnie potrzeby trzymania na dysku olbrzymich bibliotek fontów w formacie PK, wystarczą pliki źródłowe fontów i odpowiednio skonfigurowane łącze sterownik-METAFONT (program lub skrypt) by zautomatyzować proces generowania fontów. Dla bardzo rozpowszechnionych sterowników z pakietu emTeX będzie to program MFjob (MS-DOS, OS/2), zaś dla systemów opartych na web2c – skrypt MakeTeXPK (różne wersje U**X, Windows95 i NT, Amiga).

Fonty POSTSCRIPT-owe czyli pliki AFM oraz PFB (lub PFA). Jeżeli chodzi o POSTSCRIPT, to ten „właściwy font" – mowa tu o fontach zapisanych w formacie Adobe Type 1 – zawarty jest w pliku z rozszerzeniem .pfb (*POSTSCRIPT Font Binary*) lub .pfa (*POSTSCRIPT Font ASCII*). Natomiast o plikach TFM w świecie POSTSCRIPT-owym mało kto słyszał. Na szczęście dobrzy ludzie z firmy Adobe wymyślili coś podobnego do tego, co wymyślił Donald E. Knuth, mianowicie pliki metryczne AFM (*Adobe Font Metric*). Różnią się one trochę od plików TFM (np. pliki AFM są plikami tekstowymi), ale mimo różnic jest możliwe by na podstawie pliku AFM utworzyć plik TFM.

Reprezentacja znaków. Zanim pójdziemy dalej trzeba wyjaśnić jedną rzecz, mianowicie w jaki sposób są reprezentowane znaki. Jeżeli chodzi o plik DVI to odpowiedź jest trywialna: jako liczby z zakresu 0–255. Przy tworzeniu pliku POSTSCRIPT-owego za pomocą sterownika sytuacja *nie ulega* zmianie, to znaczy w pliku POSTSCRIPT-owym znaki też są reprezentowane w zasadzie jako liczby z zakresu 0–255, czasem w reprezentacji ASCII, a czasem w reprezentacji ósemkowej. Natomiast we „właściwym foncie" POSTSCRIPT-owym, podobnie zresztą jak w foncie METAFONT-owym, jest inaczej: znaki są zasadniczo reprezentowane za pomocą nazw w rodzaju period, periodcentered, lslash, aogonek, itp.

Formalnie rzecz biorąc font POSTSCRIPT-owy jest programem, a nazwy znaków są nazwami podprogramów opisujących kształt znaków za pomocą – znów podobnie jak w programach META-

FONT-owych – krzywych trzeciego stopnia, zwanych krzywymi Béziera. Dodatkowo podprogramy te opisują niektóre szczegóły procesu zamiany obwiedni na mapy bitowe (tzw. *hinty* czyli podpowiedzi), bo przecież ostatecznie znak zostanie wyświetlony na ekranie, wydrukowany na drukarce lub naświetlony na diapozytywie, czyli musi zostać zamieniony na postać dyskretną.

Nazwy i kody znaków. Skąd zatem PostScript trafiając na jakąś liczbę „wie", że ma użyć procedury o nazwie, powiedzmy, „aogonek"?

Aaa, właśnie! Tu jest pies pogrzebany. We „właściwym foncie" PostScript-owym jest zawarta dodatkowa informacja, mianowicie tabela par (liczba, nazwa) przyporządkowująca liczbom z zakresu 0 – 255 nazwy podprogramów opisujących znaki. Tablica ta (czasem zwana wektorem) nosi nazwę „Encoding", co jest może mało istotne, natomiast istotne jest to, że tablicę tę można „w locie" zmienić. Z tej możliwości korzysta m.in. sterownik DVIPS (p. niżej).

Konwersja AFM → TFM. W pliku AFM znajduje się również informacja o przyporządkowaniu liczba--nazwa, (w zasadzie można zakładać, że jest to to samo przyporządkowanie, co w pliku PFB/PFA). Na ogół nie jest to przyporządkowanie, o które by nam chodziło. Ale nie ma biedy – DVIPS-owi można podać *dowolne* przyporządkowanie, które zostanie użyte w tworzonym przezeń pliku PostScript-owym.

Konwertery AFM → TFM na podstawie nazw generują stosowne przyporządkowanie, lub nawet wiele przyporządkowań, a dla każdego z takich przyporządkowań tworzony jest odpowiadający mu plik TFM.

Najpowszechniej używanym konwerterem jest program AFM2TFM, należący do standardowej dystrybucji sterownika DVIPS. Inny konwerter o nazwie TOIL, bazujący na programach AWK oraz METAFONT, opisany jest pokrótce na stronach 22 – 23 niniejszego biuletynu. Warto też wspomnieć o pakiecie fontinst, oferującym liczne możliwości, bazującym na fontach wirtualnych (p. str. 12 – 21 niniejszego biuletynu).

Konwersja AFM → TFM – poprawność nazw znaków. Aby konwertery AFM→TFM mogły działać jak należy, nazwy podprogramów muszą być zgodne z ich treścią. Tak niestety nie zawsze jest, np. pod nazwą „Sterling" zamiast opisu znaku funta można znaleźć opis polskiego „Ł" czyli „Lslash". Pomysłów na cudaczne nazewnictwo jest mniej więcej tyle, ile tzw. standardów kodowania znaków, czyli $\approx \infty$.

Z niepoprawnymi nazwami można sobie poradzić. Co nam zależy, przecież „onesuperior" brzmi równie ładnie jak „aogonek", nieprawdaż? Problem jedynie w tym, że z góry nie wiadomo, że producent uznał „onesuperior" za odpowiednią nazwę dla polskiej litery „ą". W fontach firmy ThETA (p. niżej) akurat tak jest, ale tak bynajmniej być nie musi, w końcu jakiś szaleniec może się pogodzić z propozycją Adobe i użyć dla podprogramu opisującego „ą" nazwy „agonek"...

Jakby nie dość było problemów, oprócz nazewnictwa ewidentnie niepoprawnego możemy się zetknąć z obocznością! Otóż niektórzy producenci dla określenia znaków „ż" i „Ż" używają nazw „zdot" i „Zdot", a niektórzy „zdotaccent" i „Zdotaccent". Nie znalazłem jednoznacznego stanowiska firmy Adobe w tej sprawie, ale na ogół obowiązuje zasada „sklejania" nazw: „a" + „grave" = „agrave"; skoro więc istnieje znak „dotaccent" (nazwa używana przez Adobe), to „ż" należy rozumieć jako „z" + „dotaccent" czyli „zdotaccent". Innymi słowy zalecać by należało używanie nazw „zdotaccent" i „Zdotaccent", ale być może konwertery AFM → TFM powinny być uwrażliwione na tę oboczność.

Konwersja AFM → TFM – kerny i ligatury. Plik TFM zawiera jeszcze informację o parach kernowych i ligaturach. Pary kernowe, uwzględniane „jak leci" przez większość konwerterów AFM → TFM, na ogół są umieszczane jak należy w plikach AFM, chociaż czasami można trafić na felerne pliki AFM bez par kernowych. Obecność par kernowych poznajemy po występowaniu wierszy zaczynających się od słowa kluczowego „KPX". Warto mieć na uwadze, że bywają pliki AFM posiadające tylko część par kernowych – np. znaki diakrytyczne czasami nie są uwzględniane – dobrze jest sprawdzić plik AFM pod tym kątem.

Z ligaturami jest znacznie gorzej – można je bardzo rzadko uświadczyć, mimo iż teoretycznie format AFM pozwala na definiowanie ligatur, np. poniższy przykładowy wiersz pliku AFM (tu przełamany ze względu na ograniczoną szerokość łamu):

```
C 45; WX 329; N hyphen; B 17 188 313 303;
    L hyphen endash;
```

zgodnie ze specyfikacją Adobe ustala, że znak o kodzie 45 ma szerokość 329 jednostek (jednostka = 1/1000 em), nosi nazwę hyphen, że naroża prostokąta ograniczającego obrys znaku mają współrzędne wynoszące w tych samych jednostkach (17, 188) oraz (313, 303), i wreszcie że dwa znaki hyphen mają być zamieniane na znak o nazwie endash, czyli innymi słowy znak o nazwie endash jest ligaturą dwóch znaków o nazwie hyphen.

W związku z powszechnym brakiem informacji o ligaturach w plikach AFM, konwertery AFM → TFM muszą albo się domyślić, jakie by tu ligatury można wstawić, albo – co chyba jest lepszym rozwiązaniem – zostać poinformowane o prawidłowych dla danego fontu (grupy fontów) ligaturach.

Pliki konfiguracyjne sterownika DVIPS. Popularny sterownik DVIPS, autorstwa Tomasa Rokickiego, zamieniający pliki DVI na pliki PostScript-owe musi wiedzieć – podobnie jak każdy sterownik – gdzie znajdują się fonty TEX-owe i PostScript-owe. Informację tę zawierają pliki config.ps i psfonts.map, poszukiwane w kartotekach wskazywanych przez zmienną systemową TEXCONFIG.

Plik config.ps zawiera informację o lokalizacji plików TFM, plików PK, fontów wirtualnych, oraz fontów PostScript-owych. Natomiast plik psfonts.map zawiera informacje związane wyłącznie z dostępnymi w danej instalacji fontami PostScript-owymi. W pliku psfonts.map informacje o poszczególnych fontach umieszczane są w jednym wierszu (co czasem może być kłopotliwe). Typowy wiersz wygląda następująco:

```
qplr QuasiPalladio-Regular
     "enc-qpl ReEncodeFont"
     <qpl.enc <qplr.pfb
```

Pierwsza pozycja określa nazwę TEX-ową fontu, druga – wewnętrzną nazwę PostScript-ową fontu (robi z niej użytek interpreter PostScript-u), następnie pojawia się informacja o dodatkowych zabiegach, jakim ma być poddany font (może to być poziome przeskalowanie fontu, pochylenie lub zmiana kodowania – w tym wypadku chodzi o zmianę kodowania), a na końcu pojawiają się nazwy plików, które mają zostać włączone do wynikowaego pliku PostScript-owego – w tym wypadku jest to plik kodowania qpl.enc oraz font qplr.pfb.

Podanie nazwy pliku fontowego nie jest konieczne – w takim wypadku DVIPS założy, że chodzi o font wbudowany w urządzenie. Gorąco odradzamy korzystanie z tej możliwości – opłakane skutki korzystania z fontów wbudowanych można co jakiś czas oglądać w materiałach nawet renomowanych firm: a to zamiast polskich znaków występują „krzaczki”, a to litery zachodzą na siebie, a to skład się zupełnie rozjeżdża...

Plik kodowania zawiera tablicę 256 nazw znaków, określającą omawiane wyżej przyporządkowanie kod-nazwa. Jest to w istocie fragment kodu PostScript-owego postaci:

```
/enc-qpl[
/.notdef
/Delta
/beta
...
/thorn
/quotedblbase
] def
```

O tym, jaki kod otrzymuje dany znak, decyduje kolejność wystąpienia: w tym wypadku na pozycji zerowej (w języku PostScript liczenie zawsze odbywa się od zera) występuje znak o nazwie .notdef (znak „ciach” / w notacji PostScript-owej oznacza mniej więcej to samo co znak „w-tył-ciach” \ w notacji TEX-owej), co oznacza, że w foncie w istocie nie ma znaku o tym kodzie; znak Delta otrzymuje kod 1, beta – kod 2, ..., znak thorn – kod 254, i wreszcie znak quotedblbase – kod 255.

Warto mieć świadomość, że DVIPS włącza wszelkie pliki prawie ich nie analizując, jedyny zabieg przezeń wykonywany to zamiana postaci binarnej fontu Type 1 (PFB) na postać ASCII (PFA).

Jeżeli font jest już w postaci ASCII, to włączany jest do dokumentu „jak leci”, bez żadnej ingerencji ze strony sterownika. Oznacza to, że używając DVIPS-a możemy korzystać ze wszystkich możliwych fontów PostScript-owych, nie tylko najbardziej popularnych fontów Type 1, ale w szczególności także fontów TrueType, które dla interpretera PostScript-u są po prostu fontami Type 42 – wystarczy mieć plik TFM i font Type 42 w postaci ASCII.

Łatanie fontów – niepoprawny font. Używając różnych wymyślnych narzędzi (np. T1UTILS Lee Hetheringtona, do wzięcia z GUST-owego archiwum /fonts/utilities/t1utils.zip) można się zorientować, jakim znakom zostały przypisane jakie nazwy i w razie czego je poprawić, można też uzupełniać kerny, itp., ale to jest żmudne i przykre, bo niepotrzebne zajęcie.

Łatanie fontów – brak plików AFM. Czasem fonty PostScript-owe są dystrubuowane z plikami PFM (*Printer Font Metric*) zamiast z plikami AFM.

Brak plików AFM *praktycznie przekreśla* możliwość użycia fontu PostScript-owego w TEX-u, bowiem pliki PFM zawierają znacznie uboższą informację niż pliki AFM, w szczególności *nie zawierają* nazw znaków, czyli nie określają żadnej tablicy kodowania (*encoding vector*, p. wyżej). Można zakładać, że chodzi o takie samo przyporządkowanie, jakie jest wpisane w pliku PFB/PFA, chociaż w ogólności tak być nie musi. Ponadto w pliku PFM znajdują się dane na temat par kernowych jedynie dla co najwyżej 256 znaków, a font PostScript-owy może ich zawierać znacznie więcej (chociaż PostScript nie może wykorzystywać równocześnie więcej niż 256 znaków).

Jeśli jednak szczęście nam dopisze, to z plików PFM oraz PFB (bądź PFA) możemy odtworzyć informację niezbędną do utworzenia pliku AFM – można spróbować użyć pakietu P2A, który niedawno wzbogacił archiwum GUST (`fonts/utilities/p2a/`).

Casus ThETA. Ostatnio na rynku Polskim pojawiły się godne uwagi polonizacje fontów firmy BitStream, sygnowane przez firmę ThETA.

Niestety firma ThETA wyraźnie zignorowała rynek TEX-owy, chociaż może i słusznie zignorowała, skoro TEX-owców jakoś nie widać i nie słychać, no może z wyjątkiem kilkorga spośród 180 niegroźnych wariatów zapisanych do GUST-u...

Otóż firma ThETA wydała jak do tej pory 3 płyty CD z fontami, każda kolejna płyta zawierała inne fonty i była coraz mniej przydatna dla TEX-owców:

- Pierwsza płyta (cena ok. 500 zł) miała nazwy znaków w zasadzie poprawne z dokładnością do `zdot` zamiast `zdotaccent`. To zapewne spowodowało, że płyta ta ochrzczona została „TEX-ową” przez ludową tradycję, tworzenie plików TFM przebiegało bezboleśnie, i używanie ich ze „starym” Ghostscriptem nie wróżyło niebezpieczeństw. Niestety, fonty nie były formalnie poprawne, co powodowało, że niektóre RIP-y (interpretery PostScript naświetlarek) sygnalizowały błąd podczas ich przetwarzania; również wersja 5.10 Ghostscriptu nie akceptuje tych fontów.

  Niepoprawność formalna polegała na braku w tych fontach *wymaganego* przez dokumentację Adobe podprogramu (znaku) o egzotycznej nazwie `.notdef`. Podprogram ten jest używany, gdy z jakichś powodów program PostScript-owy odwołuje się do znaku, którego nie ma w foncie; w większości fontów jest to znak nie zawierający żadnego elementu graficznego, odpowiadający w zasadzie spacji.

  Było też drugie wydanie pierwszej płyty, „poprawione” w duchu opisanej niżej drugiej płyty.
- Z drugą płytą (cena ok. 450 zł) było lepiej jeżeli chodzi o znak `.notdef`, natomiast gorzej, jeśli chodzi o nazwy znaków. Oto errata:

JEST	WINNO BYĆ
ordmasculine	ae
ordfeminine	AE
tilde	ecircumflex
circumflex	Ecircumflex
onequarter	ntilde
logicalnot	Ntilde
c157	oe
c141	OE
cent	questiondown
florin	Ydieresis
c142	Zcaron
c158	zcaron
onesuperior	aogonek
yen	Aogonek
ae	cacute
AE	Cacute
ecircumflex	eogonek
Ecircumflex	Eogonek
threesuperior	lslash
sterling	Lslash
ntilde	nacute
Ntilde	Nacute
oe	sacute
OE	Sacute
questiondown	zdotaccent
macron	Zdotaccent
Ydieresis	zacute
c143	Zacute

  Indagowany na tę okoliczność autor polonizacji Jakub Tatarkiewicz stwierdził, że tak jest dobrze, i że nie będzie dla paru zapaleńców przygotowywał specjalnej „TEX-owej” (czyt.: poprawnej) wersji.
- Z trzecią płytą (cena ok. 300 zł) jest zdecydowanie najgorzej, bo firma ThETA nie dostarczyła plików PFM, poprzestając na plikach PFB i PFM.

Na szczęście dla TEX-owców „podłatanie” fontów ThETA nie jest trudne. W archiwum GUST-u znajduje się pakiet THETAFIX (`GUST/contrib/t1fonts/`), korzystający ze wspomnianych programów Lee Hetheringtona oraz programu AWK (w wersji Gnu), dokonujący stosownych zmian w plikach składających się na fonty PostScript-owe z dwóch pierwszych płyt. Trzecią płytę można podłatać za pomocą wspomnianego pakietu P2A.

Po takich poprawkach i utworzeniu plików TFM większość fontów powinna się dać bez kłopotów używać w dokumentach TEX-owych. Niestety większość, a nie wszystkie, bowiem kilkaset fontów z pierwszych dwóch płyt można pogrupować w kilkadziesiąt różnych klas, w zależności od zestawu znaków. Pakiet THETAFIX jest przystosowany do jednej z klas, tej najbardziej licznej. Ale w złośliwych przypadkach coś może nie zadziałać – kłopoty są ceną, jaką się płaci za tanie fonty... Tak, tak, paręset złotych za kilkaset fontów to stosunkowo tanio.

Podsumowanie. Warto pamiętać, że font musi ładnie wyglądać nie tylko w druku, ale i jego dość skomplikowana „wewnętrzna” struktura musi być zrobiona porządnie.

Jednym z najbardziej dających się we znaki problemów jest problem układu znaków w foncie. Staranne przemyślenie tego problemu *musi* prowadzić do wniosku, że firma Adobe już dawno temu znalazła *wystarczające* w większości przypadków rozwiązanie problemu konfigurowalności układu znaków w foncie. To bardzo mocny argument na rzecz korzystania z fontów POSTSCRIPT-owych w formacie Type 1.

Wprawdzie detale związane z instalowaniem fontów mogą się wydawać skomplikowane, ale taka jest natura rzeczy. A do czego prowadzi próba ukrycia tej natury przed użytkownikiem, wielu z nas boleśnie odczuło w przypadku korzystania fontów TrueType w systemie Windows. Wprawdzie instaluje się je bardzo prosto (albo i – nie wiedzieć czemu – nie instaluje), ale praktycznie nie ma *żadnej* możliwości rekonfiguracji. Ktoś, kto przenosił fonty TrueType z jednej wersji systemu Windows do innej, wie o czym mowa – a to font się w ogóle nie daje zainstalować, a to inne znaki się pojawiają, niż byśmy chcieli, a to nasza ulubiona aplikacja albo tego fontu nie widzi, albo się na nim wywraca...

Mając to na względzie uważamy, że warto poświęcić nieco czasu i wysiłku na zrozumienie podstaw konfigurowania fontów, zwłaszcza fontów POSTSCRIPT-owych w formacie Adobe Type 1 – to naprawdę w pewnym momencie staje się proste, a tym samym oczywiście niezwykle użyteczne.

◇ Bogusław Jackowski
B.Jackowski@gust.org.pl
Stanisław Wawrykiewicz
StaW@gust.org.pl

Fonty wirtualne, fonty wirtuoza[1]

Alan Hoenig

Wstęp

Dzięki fontom wirtualnym możemy w TEX-u używać wszystkich fontów, nie tylko TEX-owych, i dają nam one dodatkowe możliwości. Co to są fonty wirtualne? Kilka przykładów dostarcza nam koniecznej wiedzy na ich temat.

Artykuł stanowi fragment książki *TEX Unbound: LATEX and TEX Strategies for Fonts, Graphics, and More*. Wyjątek z książki został tak przygotowany, by można go było drukować przy użyciu standardowego zestawu fontów TEX-owych, podczas gdy oryginał posługuje się głównie fontami POSTSCRIPT-owymi i przygotowanym przez autora plikiem stylu[2]. W rezultacie artykuł nie zawiera niektórych ilustracji, obecnych w wydaniu książkowym. Wszelkie pytania i uwagi proszę przesyłać do autora na adres ajhjj@cunyvm.cuny.edu.

Kiedy mówimy o komputerach, używamy przymiotnika „wirtualny”, aby opisać coś, co zachowuje się jak coś innego. Dyski wirtualne są w rzeczywistości blokami pamięci, które symulują twardy dysk, podczas gdy pamięć wirtualna używa dysku do symulowania pamięci operacyjnej. TEX „widzi” font wirtualny tak jak każdy inny font, choć struktura fontu wirtualnego może być dość skomplikowana:

- font wirtualny może być złożony z kilku różnych fontów wymieszanych ze sobą (w specjalny sposób, zgodnie z dokładnymi przepisami);
- może to być pojedynczy font, którego znaki z jakiegoś powodu ustawione są w innej kolejności;
- może to być zbiór znaków złożonych, tj. składających się z kilku elementów (np. litery i akcentu);

1: Tłumaczenie z czasopisma TUGboat **18** (1997), No. 2, s. 113–121; Redakcja Biuletynu GUST dziękuje za zezwolenie na publikację tego artykułu.
2: Na prośbę Autora informuję, że książka się właśnie ukazała: Alan Hoenig, *TEX Unbound: LATEX and TEX Strategies for Fonts, Graphics, and More*, Oxford University Press 1997 (*StaW*).

- może to być zestaw kresek poziomych lub pionowych (`\hrule` i `\vrule`), traktowanych na prawach znaku;
- może to być zbiór plików tekstowych, graficznych lub PostScript-owych, z których każdy jest traktowany jak pojedynczy znak wewnątrz fontu wirtualnego;
- może to wreszcie być zlepek wszystkich (lub niektórych) elementów wyszczególnionych powyżej.

Niniejszy artykuł prezentuje fonty wirtualne, rozważa sytuacje, w których są potrzebne oraz dostarcza procedur do ich tworzenia. Ręczne konstruowanie fontów wirtualnych jest mało sensowne, kilka ogólnie dostępnych programów z łatwością to zrobi za nas.

Dla większości użytkowników głównym zastosowaniem fontów wirtualnych może być właściwa instalacja fontów obwiedniowych (PostScript-owych) do użytku TeX-owego (termin *instalacja* oznacza tu ciąg czynności pozwalający w efekcie na użycie fontu w TeX-u). Rezultaty trudne lub wręcz niemożliwe do osiągnięcia za pomocą makr, dają się łatwo uzyskać, jeśli zaprząc do roboty fonty wirtualne.

W kolejnych rozdziałach wyjaśniona zostanie szczegółowo koncepcja fontu wirtualnego oraz omówione zostaną istotne w tym kontekście pojęcia tabel fontów i tabel kodowania. Następnie przedstawione zostaną aplikacje wspomagające zastosowanie fontów wirtualnych.

1. Koncepcja fontu wirtualnego

Zacznijmy od podróży na inną planetę, taką, na której pojawił się system podobny do TeX-a, ale wszystkie języki zawierają jedynie dwa różne znaki, nazwane „e” i „f” oraz ligaturę „ff”. Baczne przyjrzenie się fontowi z hipotetycznej planety ułatwi zrozumienie problemów pojawiających się w wypadku rzeczywistych, ziemskich fontów i sposobu, w jaki fonty wirtualne mogą je rozwiązać.

Tabela podająca listę znaków jakiegokolwiek fontu TeX-owego zawierałaby tylko trzy znaki.

0	1	2
e	f	ff

Znakom zostały przypisane liczby według zwykłej informatycznej konwencji, poczynając od zera. Liczbowe oznaczenia (kody) służą identyfikacji pozycji każdego znaku w foncie. Mają one też istotne znaczenie dla TeX-a, ponieważ pliki `dvi` zawierają polecenia składania oparte *nie* na nazwie obrysu znaku (ang. *glyph*): „A”, „B”, „comma” itp., ale na liczbowej etykiecie. Dla każdego „e” w pliku wejściowym, plik `dvi` zawiera instrukcję składu znaku 0 w bieżącym foncie. Lepiej więc, żeby małe „e” znajdowało się w tej pozycji! Zależność między znakiem i kodem liczbowym znaku jest wbudowana w program TeX; przyjmiemy, że jest to prawdą zarówno na odległej planecie, jak i na Ziemi.

Kiedy spróbujemy użyć fontu komercyjnego zamiast fontu rodziny Computer Modern, możemy spodziewać się trudności. Przypuśćmy, że font komercyjny, którego chcemy użyć, zawiera wprawdzie trzy znaki, z tym że są to „e”, „f” i niepotrzebny nam znak „&”. Aby móc uzyskać ligaturę, musimy zakupić dodatkowy font, który zawiera znak „ff” i oprócz tego dwa zupełnie zbyteczne znaki. Układ znaków w obu fontach przysparza dodatkowych kłopotów:

0	1	2
f	e	&

0	1	2
ff	%	$

Znaki w foncie głównym są ustawione w innej kolejności, niż poprzednio, a to prowadzi do katastrofy. Żeby zobaczyć dlaczego, wybierzmy font komercyjny i napiszmy „f”. TeX oczekuje, że „f” będzie w pozycji 1 w tabeli fontów i dlatego w pliku `dvi` umieści informację, że złożony ma być znak o kodzie 1. Ale znak o tym kodzie w foncie nie-TeX-owym oznacza obrys „e” i tak też zostanie złożony, zamiast pożądanego „f”. Ponadto okazuje się, że nie możemy złożyć „ff” bez jawnego odwołania się do fontu pomocniczego. Krótko mówiąc plik wejściowy będzie wyglądał inaczej, gdy złożymy go zwykłym fontem TeX-owym, a inaczej gdy złożymy go innymi fontami. Jest to sytuacja nie do przyjęcia.

Fonty wirtualne zostały stworzone po to, aby radzić sobie z tego rodzaju trudnymi sytuacjami. Z punktu widzenia użytkownika font wirtualny jest takim fontem jak inne. Za kulisami działa jednak mechanizm, za pomocą którego „rzeczywiste” fonty, zwane dalej *fontami pierwotnymi – raw fonts*, mogą być zestawione w taki sposób, że wynikowy *font wirtualny* zachowuje się – z punktu widzenia TeX-owca – jak zwykły font. Eliminuje to potrzebę stosowania specjalnej notacji w pliku wejściowym.

0 Γ *0*	1 Δ *1*	2 Θ *2*	3 Λ *3*	4 Ξ *4*	5 Π *5*	6 Σ *6*	7 Υ *7*
8 Φ *10*	9 Ψ *11*	10 Ω *12*	11 ff *13*	12 fi *14*	13 fl *15*	14 ffi *16*	15 ffl *17*
16 ı *20*	17 ȷ *21*	18 ` *22*	19 ´ *23*	20 ˇ *24*	21 ˘ *25*	22 ¯ *26*	23 ˚ *27*
24 ¸ *30*	25 ß *31*	26 æ *32*	27 œ *33*	28 ø *34*	29 Æ *35*	30 Œ *36*	31 Ø *37*
32 - *40*	33 ! *41*	34 ” *42*	35 # *43*	36 $ *44*	37 % *45*	38 & *46*	39 ’ *47*
40 (*50*	41) *51*	42 * *52*	43 + *53*	44 , *54*	45 - *55*	46 . *56*	47 / *57*
48 0 *60*	49 1 *61*	50 2 *62*	51 3 *63*	52 4 *64*	53 5 *65*	54 6 *66*	55 7 *67*
56 8 *70*	57 9 *71*	58 : *72*	59 ; *73*	60 ¡ *74*	61 = *75*	62 ¿ *76*	63 ? *77*
64 @ *100*	65 A *101*	66 B *102*	67 C *103*	68 D *104*	69 E *105*	70 F *106*	71 G *107*
72 H *110*	73 I *111*	74 J *112*	75 K *113*	76 L *114*	77 M *115*	78 N *116*	79 O *117*
80 P *120*	81 Q *121*	82 R *122*	83 S *123*	84 T *124*	85 U *125*	86 V *126*	87 W *127*
88 X *130*	89 Y *131*	90 Z *132*	91 [*133*	92 “ *134*	93] *135*	94 ˆ *136*	95 ˙ *137*
96 ‘ *140*	97 a *141*	98 b *142*	99 c *143*	100 d *144*	101 e *145*	102 f *146*	103 g *147*
104 h *150*	105 i *151*	106 j *152*	107 k *153*	108 l *154*	109 m *155*	110 n *156*	111 o *157*
112 p *160*	113 q *161*	114 r *162*	115 s *163*	116 t *164*	117 u *165*	118 v *166*	119 w *167*
120 x *170*	121 y *171*	122 z *172*	123 – *173*	124 — *174*	125 ˝ *175*	126 ˜ *176*	127 ¨ *177*

Rys. 1. Tabela fontów Computer Modern Roman (tutaj, cmb10). Liczby pisane krojem prostym w lewej części każdej komórki tabeli podają kod dziesiętny znaku; z prawej strony podany jest kursywą kod w notacji ósemkowej.

W naszym wypadku font wirtualny miałby do zrealizowania następujące zadanie:

1. wybrać „e” i „f” z głównego fontu i zmienić stosownie ich szyk;
2. włączyć ligaturę z fontu eksperckiego w ostatniej pozycji w tabeli fontu wirtualnego.

Mianem fontu eksperckiego określa się font dodatkowy, zawierający ligatury, inicjały i inne specjalne symbole.

Jeśli chodzi o nazewnictwo, to – zgodnie z ziemską konwencją – fontom pierwotnym nadajemy nazwę z przyrostkiem „8a”, fontom eksperckim – nazwę z przyrostkiem „8x”, natomiast fonty wirtualne otrzymują przyrostek „7t”. Jeśli więc font nosi nazwę `foo`, to font pierwotny powinien się zwać `foo8a`, font ekspercki – `foo8x`, a utworzony z nich font wirtualny – `foo7t` (użyta tu konwencja nazwnictwa pochodzi od Karla Berry'ego; jej wyjaśnienie można znaleźć w jakimkolwiek archiwum CTAN w obszarze `info/fontname`):

0 1 2		0 1 2		0 1 2
f e &	+	ff % $	⇒	e f ff
`foo8a`		`foo8x`		`foo7t`

Font `foo7t` używa wybranych znaków z dwóch fontów pierwotnych i wybór ten jest ustawiany w szyku mającym znaczenie dla TEX-a. Nie wszystkie znaki fontu pierwotnego muszą być częścią końcowego fontu wirtualnego.

Do czasu pojawienia się fontów wirtualnych słowa zawierające znaki akcentowane były ignorowane przez TEX-owy algorytm dzielenia wyrazów. W foncie wirtualnym możemy zdefiniować literę akcentowaną tak, aby „zachowywała się” jak zwykła litera, tzn. tak, by algorytm dzielenia wyrazów działał bez zakłóceń – to kolejna zaleta fontów wirtualnych.

Ten pozornie nierealistyczny przykład ilustruje problemy, z jakimi przychodzi borykać się w praktyce. W zastosowaniach praktycznych dodatkową komplikację stanowi duża liczba znaków w fontach – trzy znaki w foncie to drastyczne ułatwienie i raczej nie należy spodziewać się od losu takich ułatwień.

2. Fonty i tabele fontów

Tabele TEX-owych fontów Computer Modern i obwiedniowych fontów POSTSCRIPT-owych zawierają maksymalnie 256 pozycji (256 jest jedną z magicznych liczb w informatyce). Spójrz na rys. 1 – przykład tabeli fontów rzeczywistych (nieco inny sposób prezentacji zastosowano w tabelach fontów zaczynających się na stronie 427 *The TEXbook*).

Tabele fontów POSTSCRIPT-owych z reguły wyglądają odmiennie. Np. każda komórka tabeli fontu Computer Modern jest wypełniona (aż do znaku 127), podczas gdy w wypadku fontów POSTSCRIPT-owych wiele komórek jest pustych. Bywa, że niektóre znaki, jak w naszym przykładzie greckie litery lub ligatury „ff", „ffi" i „ffl", nie pojawiają się wcale w fontach POSTSCRIPT-owych. Z kolei pewnych znaków obecnych w fontach POSTSCRIPT-owych brak w zestawie Computer Modern. Niektóre znaki pojawiają się i tu, i tu, ale w innych pozycjach, np. ligatury skandynawskie („æ", „œ" itp.) mają w fontach Computer Modern kody niższe od 32, natomiast w fontach POSTSCRIPT-owych – o ile w ogóle występują – mają kody wyższe niż 127.

Pozycje w tabelach znaków na ogół są numerowane poczynając od zera i kończąc na 255. Wiemy, że „ziemski" TEX wybiera znaki *nie* według nazwy znaku, ale zgodnie z jego pozycją w tabeli fontów, tak więc polecenie składu „A" zostaje przekazane do pliku dvi jako instrukcja

złóż znak z bieżącego wybranego fontu,
który zajmuje pozycję 65 w tym foncie

ponieważ jest to liczbowe oznaczenie pozycji „A".

Kiedy używa się POSTSCRIPT-owych fontów obwiedniowych, dobrze jest móc składać „fałszywym fontem" – naszym fontem wirtualnym – który dla TEX-a wygląda jak rzeczywisty, ale faktycznie jest amalgamatem jednego lub kilku pierwotnych, złożonych fontów. Ten wirtualny font organizujemy tak, aby znaki były w takim samym porządku jak w każdym innym foncie Computer Modern. Dzięki temu makra rzadko będą musiały być przedefiniowywane dla różnych fontów, co jest szczególnie ważne przy składzie matematyki.

Z tabelą fontów związany jest wektor kodowania (*font encoding vector*), zwany też stroną kodową (*code page*). Wektor kodowania jest listą nazw znaków ułożoną w porządku występowania w tabeli fontów. W wypadku fontu TEX-owego (rys. 1) wektor kodowania jest listą, której początek wygląda następująco:

```
Gamma, Delta, Theta, Lambda, Epsilon,
Pi, Sigma, Upsilon, Phi, Psi, Omega,
ff, fi, fl, ffi, ffl, dotlessi, ...
```

Jeśli umówimy się, że „.notdef" będzie oznaczać pozycję fontu, dla której żaden znak nie jest zdefiniowany, wtedy w wypadku fontu POSTSCRIPT-owego wektor kodowania jest listą, której początek (na ogół) prezentuje się zupełnie inaczej:

```
.notdef, ..., .notdef, (32 razy)
space, exclam, quotedbl, ...
```

3. Co zawiera font wirtualny?

TEX potrafi przetwarzać jedynie znaki występujące w metryce danego fontu. Oczekuje, że znajdzie tę informację w pliku tfm, tak więc każdy font wirtualny musi być zaopatrzony w plik metryczny. Plik ten powinien znajdować się w odpowiednim dla danej instalacji miejscu.

Szczegóły konstrukcji znaków wirtualnych opisane są w pliku fontu wirtualnego – pliku z rozszerzeniem .vf. Na twardym dysku powinny być wydzielone obszary (katalogi) do przechowywania fontów wirtualnych, analogicznie do obszarów wydzielonych na pliki tfm, formaty, makra itd. Obszary te noszą różne nazwy w zależności od tego, czy nasz system jest zgodny ze standardem TDS, czy też stosujemy swoje prywatne nazewnictwo.

Plik fontu wirtualnego vf zawiera w istocie fragmenty języka dvi, które określają sposób, w jaki znak wirtualny powinien być utworzony. Oznacza to, że znak w foncie wirtualnym może być niemalże dowolnym fragmentem pliku dvi. Teoretycznie jeden znak wirtualny może złożyć całą stronę dokumentu! Jednak w praktyce znaki wirtualne nie są aż tak złożone. W przykładzie z obcą planetą font wirtualny po prostu połączył dwa fonty w jeden, przypisując zarazem znakom inne kody.

4. Czego będziemy potrzebować – przygotowanie

Wielu autorów będzie przygotowywać dokumenty dla POSTSCRIPT-owych urządzeń drukujących. Ponieważ TEX wie jedynie jak napisać plik dvi, zawsze będziemy potrzebować konwertera dvi → POSTSCRIPT. Czasami programy te wymagają pomocniczego pliku mapowego, aby móc odwzorować długie (wewnętrzne) nazwy fontów POSTSCRIPT-owych na krótkie nazwy plików fontowych, akceptowane przez niektóre proste systemy operacyjne, takie jak MS-DOS i jemu podobne.

Konwerter dvi → PostScript o nazwie dvips jest dostępny na prawie wszystkich platformach komputerowych, dlatego będziemy w dalszym ciągu odwoływać się do niego i do jego pliku mapowego psfont.map. Każdy wiersz pliku psfont.map zawiera krótką, akceptowaną przez DOS nazwę pliku fontu *pierwotnego* oraz jego długą nazwę PostScript-ową. Te krótkie aliasy są nazwami, których powinniśmy używać w procesie tworzenia fontów wirtualnych. Dla każdego krótkiego aliasu z pliku mapowego musimy posiadać plik tfm o dokładnie tej samej nazwie – duże i małe litery są w tym kontekście istotne.

Plik mapowy może pełnić inne funkcje, np. pomagać w procesie ładowania (*downloading*) fontów do urządzenia (p. niżej), może też określać proste transformacje fontu (zwężenie, poszerzenie i pochylenie).

W jaki sposób drukarka otrzymuje informacje o kształtach znaków? W wypadku fontów bitmapowych sterownik drukarki, czyli program zamieniający plik dvi na język danej drukarki, jest odpowiedzialny za przekazanie drukarce informacji o mapach bitowych, reprezentujących kształty poszczególnych znaków. Sytuacja jest inna w wypadku fontów obwiedniowych, (zwanych też niezbyt ściśle fontami wektorowymi lub fontami skalowalnymi). Tym razem informacja o obwiedniach wszystkich znaków w foncie musi być przekazana drukarce, ponieważ ostatecznie to drukarka PostScript-owa w trakcie drukowania dokonuje konwersji obwiedni na postać rastrową (pikselową). Definicje trzydziestu pięciu standardowych fontów PostScript-owych, włączając popularne fonty *Times Roman* i *Helvetica*, są wbudowane w większości drukarek rozumiejących PostScript. Jeśli użyje się innych fontów, niż te znajdujące się w drukarce, trzeba będzie je *załadować* – przesłać informację o fontach do drukarki, i może to być dokonane w różny sposób. Możliwe jest także włączenie informacji o foncie do PostScript-owej wersji dokumentu.

5. Wariant podstawowy: afm2tfm

Na szczęście problem konstruowania fontu wirtualnego z fontu PostScript-owego nie jest taki beznadziejny. Większość znaków znajduje się w tej samej pozycji, włączając małe i duże litery, cyfry i znaki interpunkcyjne. Pozostałe znaki w foncie obwiedniowym możemy podzielić na dwie grupy:

- znaki specjalne, takie jak „fi", „—", „¿", „¡" czy cudzysłowy, do których dostęp realizowany jest za pomocą TEX-owego mechanizmu ligaturowego;
- znaki takie jak æ, Œ lub ç, które są wywoływane przez sekwencje sterujące TEX-a (*control sequences*), tutaj \ae, \OE i \c{c}.

(Istnieje też trzecia grupa – znaki występujące w fontach rodziny Computer Modern, ale nieobecne w typowych fontach PostScript-owych; takimi znakami są na przykład potrójne ligatury „ffi" oraz „ffl". Dalej zobaczymy, jak sobie z tym problemem radzić.)

Chcielibyśmy mieć dostęp do tych znaków fontu bez konieczności zmiany zasad, według których tworzymy nasz dokument źródłowy. Dotyczy to także przypadków, gdy autor jest uparty i używa niestandardowej konwencji do składania symboli (np. symbol „¿" zapisuje jako \char62 lub \symbol{62} zamiast ?‘); chcielibyśmy, aby układ fontu wirtualnego był zbliżony jak tylko się da do układu oryginalnego fontu TEX-owego.

Ligatury pierwszej grupy mogą być osiągnięte w sposób niewirtualny poprzez dostosowanie plików metrycznych, tak aby TEX wybierał ligaturę z właściwej pozycji fontu; nie jest wtedy potrzebne ponowne mapowanie. Wystarczy jedynie modyfikacja pliku tfm.

Znaki dostępne za pomocą poleceń TEX-a wymagają więcej zachodu. Definicja każdego takiego polecenia zależy od pozycji specjalnych znaków w tabeli fontu. TEX oczekuje więc, że „œ" będzie znakiem 27 w foncie, ponieważ znajduje się na tej pozycji w rodzinie fontów Computer Modern. Kiedy tworzy „ç", oczekuje – z powodu jak wyżej – że cedilla będzie w pozycji 24. Zazwyczaj znaki te nie pojawiają się na tych pozycjach w PostScript-owych fontach pierwotnych, na przykład w tzw. *Adobe Standard Encoding* „œ" i cedilla zajmują pozycje 250 i 203. Makra można przedefiniować, ale uzależnianie definicji makr od bieżącego fontu to zły pomysł. Zastosowanie fontu wirtualnego pozwala na zmianę porządku, na przemapowanie tych znaków w foncie – znak o kodzie 27 fontu wirtualnego można utworzyć ze znaku o kodzie 250 z fontu pierwotnego. Tym samym gdy font wirtualny jest fontem bieżącym, makro \oe będzie prawidłowo składać obrys „œ".

Programu afm2tfm używamy do zainstalowania fontu wówczas, gdy pojedynczy font pierwotny jest jedynym komponentem fontu wirtualnego. Przyjmując, że posiadamy plik foo8a.afm i jeden z plików foo8a.pfb, foo8a.pfa lub foo8a (w zależności od systemu komputerowego i sposobu dystrybucji fontów), tworzymy font wirtualny foo7t według następującej procedury:

1. uruchomić polecenie
   ```
   afm2tfm foo8a.afm -v foo7t.vpl foo8a.tfm >tmp
   ```
2. uruchomić polecenie
   ```
   vptovf foo7t.vpl foo7t.vf foo7t.tfm
   ```
3. plik tmp zawiera pojedynczy wiersz
   ```
   foo8a PełnaNazwaFontu
   ```
   który należy dodać do pliku psfonts.map, jeśli chcemy, aby dvips ładował automatycznie plik fontu, należy zmodyfikować zapis do postaci
   ```
   foo8a PełnaNazwaFontu </psfonts/foo8a.pfb
   ```
4. przenieść oba pliki .tfm, jeden plik .vf oraz oryginalne pliki fontu do odpowiednich katalogów; należy poza tym sprawdzić, czy psfonts.map jest tam, gdzie powinien.

Rys. 2. Tworzenie fontu wirtualnego za pomocą afm2tfm.

Program afm2tfm (część pakietu dvips) jest wspaniałym narzędziem do tworzenia podstawowego rodzaju fontu wirtualnego – fontu powstałego z przemapowania znaków w pojedynczym foncie pierwotnym. Ponieważ źródło tego programu jest dostępne, istnieją jego skompilowane wersje praktycznie dla każdej istotnej platformy komputerowej i pliki binarne są ogólnie dostępne u przyjaciół lub w archiwach (to samo dotyczy sterownika dvips). Program afm2tfm ma jednak kilka wad: nie może tworzyć fontu wirtualnego z więcej niż jednego pliku pierwotnego, nie może utworzyć deskryptorów fontów fd, których używa LaTeX, i nie naśladuje tak dobrze, jak by mógł, oryginalnego układu fontu TeX-owego. Niemniej jednak, proste instalacje fontów PostScript-owych z wykorzystaniem programu afm2tfm są na tyle istotne, że warto dokładniej opisać proces.

Schemat procedury tworzenia fontów wirtualnych w przypadku najprostszym przedstawiony jest na rysunku 2 (do bardziej skomplikowanych instalacji lepiej użyć pakietu *fontinst* – p. niżej). W procedurze tej pojawiają się pliki różnych typów. Jeśli obwiedniowym fontem jest np. psfont, to oznacza, że dystrybucja powinna zawierać psfont.afm i psfont.pfb. Konieczna jest zmiana nazwy fontu na psfont8a, zaś wygenerowane pliki – nasz font wirtualny – będą miały nazwy psfont7t.vpl, psfont7t.tfm i psfont7t.vf. Generowany jest również plik metryczny psfont8a.tfm dla pierwotnego fontu PostScript-owego.

Program afm2tfm może również tworzyć pseudokapitaliki, a także odmiany powstające przez poddanie fontu pierwotnego prostym geometrycznym transformacjom, takim jak pochylenie (*slanting*) lub rozszerzenie (*extension*). (Sprawdź dokumentację, aby dowiedzieć się, jak to się robi.)

Gdy już utworzymy font wirtualny i umieścimy wszystkie pliki tam, gdzie powinny być, możemy korzystać z fontów wirtualnych tak jak ze zwykłych fontów, na przykład możemy zadeklarować font w następujący „plainowy" sposób:

```
\font\foo=psfont7t at 10.5pt
```

i użyć go za pomocą polecenia zmieniającego font, tzn. \foo, analogicznie jak \it lub \tt. Nie korzystamy tu jawnie (TeX zresztą też nie korzysta) z fontu pierwotnego, dopiero sterowniki TeX-owe dokonują pełnej analizy danych zawartych w plikach tworzących font wirtualny i w razie potrzeby odwołują się do komponentów, czyli fontów pierwotnych.

6. Wariant rozbudowany: *fontinst*

Pakiet *fontinst* Alana Jeffreya wykonuje wszystko to, co afm2tfm, a nawet więcej. Na przykład, może utworzyć font wirtualny z kilku fontów pierwotnych i automatycznie generuje posiłkowy plik fd, używany przez LaTeX-owy mechanizm NFSS przełączania fontu. Pakiet *fontinst* jest napisany całkowicie w TeX-u. Wielbicielom TeX-a powinno sprawić radość czytanie fontinst.sty – można dowiedzieć się, jak TeX robi rzeczy, do których nie był w zasadzie przeznaczony. Napisanie pakietu w języku TeX-owym jest gwarancją, że *fontinst* działa na każdej platformie, na której działa TeX. Pakiet *fontinst* można skopiować z dowolnego archiwum CTAN, z katalogu fonts/utilities/. Niniejszy artykuł stanowi uzupełnienie `fontinst.tex` – dokumentacji pakietu.

Pakietu *fontinst* używamy przez przygotowanie prostego pliku TeX-owego w formacie `plain`. Plik ten będzie krótki i będzie zawierał polecenie `\input fontinst.sty`, po którym nastąpią inne polecenia, precyzujące jak utworzyć font wirtualny. Ponieważ pliki `vf` i `tfm` są plikami binarnymi, a TeX potrafi tworzyć jedynie pliki tekstowe (nie licząc pliku `dvi`), *fontinst* czyta i pisze pliki będące tekstowymi odpowiednikami niezbędnych plików binarnych, tzw. listy właściwości, np. pliki z rozszerzeniem `.vpl` i `.pl`, zawierające listy właściwości dla fontu wirtualnego i metryki TeX-owej, są odpowiednikami ASCII plików `.vf` i `.tfm`. TeX-owa instalacja powinna zawierać programy narzędziowe `vptovf` i `pltotf`, służące do zamiany list właściwości na postać binarną, oraz programy `vftovp` i `pltotf` o działaniu odwrotnym. Narzędzi tych użyjemy do utworzenia potrzebnych nam plików fontowych.

Po poprawnym zadziałaniu *fontinst*, w bieżącym katalogu zostaną wygenerowane pliki trzech rodzajów:

- pliki `pl`, po jednym dla każdego fontu pierwotnego; wykorzystane zostaną przez program `pltotf` do generowania pliku `tfm`;
- `vpl`, po jednym dla każdego fontu wirtualnego; wykorzystane zostaną przez program `vptovf` do generowania pliku `vf` i pliku `tfm`;
- deskryptor `fd` dla każdej rodziny fontów, wykorzystywany przez LaTeX-owy mechanizm NFSS do opisu relacji atrybutów indywidualnych fontów.

(Pojawiają się również pewne nowe pliki `mtx` i oczywiście plik `log`, który można usunąć.) Konieczne jest przetworzenie wszystkich plików `vpl` przez `vptovf` i wszystkich plików `pl` przez `pltotf`, aby wygenerować potrzebne dla TeX-a binarne pliki fontowe. Plik mapowy, taki jak `psfonts.map` dla `dvips`, powinien zostać uaktualniony.

Wszystkie pliki `tfm` należy umieścić tam, gdzie przechowujemy pozostałe pliki `tfm`. Specjalne miejsce należy również przeznaczyć na pliki `vf`, poinformowawszy oczywiście `dvips`-a, gdzie ma szukać fontów wirtualnych. Pliki `fd` umieszczamy tam, gdzie trzymamy pakiety makr TeX-owych.

7. Instalowanie pakietu *fontinst*

W skład pakietu *fontinst* wchodzi plik `fontinst.sty`, dokumentacja, pliki dodatkowe, a także wiele przykładów.

Pakiet jest dystrubuowany w postaci archiwum `zip` ze strukturą katalogów. Wygodnie jest utworzyć katalog `.../fontinst`, w którym będziemy rozpakowywać archiwum. W podkatalogu `inputs` poza samym `fontinst.sty` znajdą się jeszcze pliki z rozszerzeniami `.mtx` i `.etx`. Aby zakończyć instalację, należy przenieść te pliki do jednego z TeX-owych katalogów zawierających makra. Pakiet *fontinst* dostarcza nowego języka do opisu procedur tworzenia fontów wirtualnych dowolnego typu. Warto starannie przyjrzeć się tym procedurom – to pozwoli nam zainstalować dowolny font przy (stosunkowo) niewielkim wysiłku.

8. Pakiet *fontinst* – podstawy

8.1. Nowe polecenia

Na rys. 3 pokazano jeden ze sposobów użycia *fontinst* do zainstalowania fontów *Times Roman*, które są wbudowane w każdą drukarkę PostScript-ową.[3] Większość plików instalacyjnych pakietu *fontinst* ma analogiczną postać.

Treść przykładu z rys. 3 nietrudno zrozumieć. Pierwszy wiersz

```
\input fontinst.sty
```

wczytuje makra. Para poleceń `\installfonts` i `\endinstallfonts` (bez argumentów) otacza sekwencję poleceń, które wykonują większość pracy. Pierwszy argument poleceń `\installfamily` określa kodowanie, drugi – przypisanie do rodziny fontów a trzeci – zbiór poleceń, które będą wykonywane za każdym razem, kiedy rodzina jest ładowana:

```
\installfamily{encoding}{family}%
      {fd-commands}
```

Dalszych szczegółów na temat trzeciego argumentu należy szukać w dokumentacji *fontinst*; argument ten przeważnie będzie pusty.

Głównym poleceniem w dowolnym pliku instalującym jest polecenie `installfont`, które ma 8 parametrów. Ostatni parametr specyfikuje rozmiar

3: Redakcja Biuletynu czuje się w obowiązku przestrzec użytkowników polskich przed korzystaniem z fontów wbudowanych – chyba że ktoś lubi kłopoty...

```
\input fontinst.sty
\installfonts
   \installfamily{OT1}{ptm}{}
   \installfont{ptmr7t}{ptmr8a,latin}%
      {OT1}{OT1}{ptm}{m}{n}{}
   \installfont{ptmrc7t}{ptmr8a,latin}%
      {OT1c}{OT1}{ptm}{m}{sc}{}
   \installfont{ptmri7t}{ptmri8a,latin}%
      {OT1}{OT1}{ptm}{m}{it}{}
   \installfont{ptmb7t}{ptmb8a,latin}%
      {OT1}{OT1}{ptm}{bx}{n}{}
   \installfont{ptmbc7t}{ptmb8a,latin}%
      {OT1c}{OT1}{ptm}{bx}{sc}{}
   \installfont{ptmbi7t}{ptmbi8a,latin}%
      {OT1}{OT1}{ptm}{bx}{it}{}
\endinstallfonts
\bye
```

Rys. 3. Jeden ze sposobów instalowania fontu *Times Roman*. Użyte zostało oryginalne TEX-owe kodowanie, bez fontów eksperckich. Zainstalowane będą dwa szeregi: regularny i pogrubiony. W każdym szeregu – trzy odmiany: prosta, kapitalikowa (używająca pliku kodowania OT1c.etx) i kursywa.

fontu (stopień pisma); w wypadku fontów obwiedniowych jest prawie zawsze pusty, ponieważ fonty te dają się skalować w dowolny sposób. (Fonty bitmapowe, tworzone dla różnych rozmiarów, wymagają parametru niepustego.) Parametry od czwartego do siódmego zawierają kolejno: kodowanie, rodzinę, serię i krój pisma fontu, używane przez *fontinst* do utworzenia pliku fd dla NFSS. Pierwszy parametr przechowuje nazwę pliku fontu wirtualnego, który chcemy utworzyć. Pozostają parametry drugi i trzeci. Aby zrozumieć ich znaczenie, musimy zrobić małą dygresję na temat procesu tworzenia fontu.

8.2. Tworzenie fontów

Istnieją dwa aspekty tworzenia fontów:

- **Metryczny:** Potrzebujemy procedur do konstrukcji (w sensie wirtualnym) każdego obrysu lub znaku fontu wirtualnego. Używam dalej terminu „metryczny", aby objąć te aspekty strukturalne, które ostatecznie wymagają mierzenia obrysów dla potrzeb TEX-a.
- **Kodowanie:** Musimy zdecydować, jaki jest porządek znaków w foncie i określić wszelkie dodatkowe reguły potrzebne do użycia znaku. Na przykład, mogą one dotyczyć ligatur (za każdym razem, gdy „i" następuje po pojedynczym „f", jest zastępowane przez „fi"; podobnie, gdy „A" pojawia się na początku słowa, jest zastępowane przez odpowiednio dobrany wariant obrysu) lub symboli matematycznych (za każdym razem, gdy rozmiar nawiasu zostaje dobrany do wysokości składanego materiału).

Instrukcje te powinny znaleźć się w *pliku metrycznym* pakietu *fontinst* z rozszerzeniem mtx i w *pliku kodowania* z rozszerzeniem etx. W drugim parametrze polecenia \installfont umieszczamy listę plików metrycznych, które powinny zostać uwzględnione podczas przetwarzania; *fontinst* czyta je, aby dowiedzieć się jak tworzyć znaki. Trzeci parametr zawiera nazwę pliku kodowania, określającego, które znaki należy włączyć do fontu, w jaki sposób je uporządkować, jakie występują ligatury, oraz inne szczególne reguły. Oto postać instrukcji \installfont:

```
\installfont{font-name}{metric-files}%
   {encoding-file}{encoding}%
   {family-name}{series}{shape}{size}
```

8.3. Pliki metryczne

Przygotowanie plików metrycznych dla TEX-a jest ułatwione, ponieważ *fontinst* czyta trzy typy plików metrycznych:

- pliki mtx, formatu specyficznego dla *fontinst*;
- pliki afm, pliki metryczne ASCII, które dostarczane są z każdym przyzwoitym fontem PostScript-owym;
- pliki pl, odpowiedniki ASCII plików tfm.

fontinst czyta dwa pierwsze typy automatycznie; do przetworzenia trzeciego typu pliku wymagany jest programu tftopl – powinien on występować w instalacji TEX-a. Żeby się o tym przekonać wystarczy sprawdzić, czy zostanie wykonane polecenie:

```
tftopl cmr10.tfm cmr10.pl
```

W plikach konfiguracyjnych *fontinst* obowiązuje kolejność występowania definicji, dokładniej: jeżeli jakaś konstrukcja pojawia się kilkakrotnie w serii plików czytanych przez *fontinst*, to tylko pierwsza z nich jest brana pod uwagę; późniejsze definicje są milcząco ignorowane. Wobec tego, *szyk, w jakim fontinst czyta pliki, jest bardzo ważny!* Jak zobaczymy, jest to najważniejsza cecha sposobu pracy *fontinst*.

Plik latin.mtx jest „plikiem metrycznym ostatniej instancji" – dostarcza instrukcji do tworzenia 401 kształtów zawartych w alfabetach łacińskich.

Z tych 401 kształtów niektóre są nie do podrobienia – nie ma sposobu wydrukowania znaku takiego jak „A" jeśli znaku o tym kształcie brakuje w foncie, ale wiele innych obrysów można „podrobić". Litery akcentowane mogą być zbudowane z liter i akcentów, minuskuły kapitalików mogą być wzięte ze zbioru wersalików fontów przeskalowanych do 80% bieżącej wielkości projektowej. Rzecz jasna dla wszystkich 401 znaków nie starczy miejsca w pojedynczym foncie, nieprzekraczalną granicą jest liczba 256. Ponieważ wiele z tych znaków zostało wcześniej zdefiniowanych w plikach metrycznych, *fontinst* będzie ignorował niektóre definicje z `latin.mtx`, gdyż – jak to zostało odnotowane – liczy się pierwsze wystąpienie danej definicji. Jeśli znak, do którego się następnie odwołujemy, nie został zdefiniowany, to definicje z `latin.mtx` posłużą jako „bezpiecznik". Z tego właśnie powodu wszystkie polecenia `\installfont` z rys. 3 i w każdym przykładzie pakietu *fontinst* zawierają listy plików metrycznych, które kończą się odwołaniem do `latin.mtx`.

8.4. Pliki kodowania

Pliki metryczne opisują konstruowanie znaków, natomiast pojedynczy plik kodowania wybiera grupę znaków przynależną do fontu i określa właściwą ich kolejność (*encoding*). Plik ten określa również pewne ligatury i inne reguły, jakim podlegać ma font.

Nazwy plików kodowania odzwierciedlają kodowanie. Tak więc plik opisujący kodowanie `OT1` ma nazwę po prostu `OT1.etx`. Podobnie pliki `OT1c.etx` i `OT19.etx` określają porządek fontów kapitalikowych i fontów *old-style* z wykorzystaniem kodowania `OT1`. W dystrybucji *fontinst* występują jeszcze inne warianty.

9. Kolejne przykłady

Najwyższy czas przyjrzeć się przykładom użycia *fontinst* do tworzenia fontów wirtualnych.

9.1. Prosta instalacja fontów

Najprostszym sposobem użycia *fontinst* jest uruchomienie TEX-a z plikiem `fontinst.sty` i w odpowiedzi na TEX-owy komunikat * napisanie

```
\latinfamily{ptm}{}
\bye
```

Zakładamy, że wszystkie fonty w rodzinie `ptm` (*Times Roman*) zostały nazwane według schematu Karla Berry'ego i że wszystkie pliki metryczne fontów znajdują się w miejscach przeszukiwanych przez TEX-a.

Metoda ta jest najlepsza dla autorów, którzy uważają, że nigdy nie będą potrzebować bardziej egzotycznych fontów. Kolejne przykłady są tak pomyślane, aby pokazać siłę *fontinst* i wyjaśnić jego zawiłości w poglądowy sposób.

9.2 Proste fonty EC

Kodowanie Cork, oznaczone przez `T1`, odnosi się do standardu przyjętego na konferencji TEX-owej w Cork (Irlandia), we wrześniu 1990 roku. Osiągnięto wówczas porozumienie dotyczące układu fontów o 256 znakach na użytek TEX-a. (Nieco wcześniej TEX został przystosowany do obsługi fontów 256-znakowych.) Fonty `ecr` wyglądają jak zwykłe fonty Computer Modern, ale zostały one rozszerzone zgodnie ze standardem z Cork. Fonty wirtualne zapewniają prosty sposób generowania fontów `ecr` z podstawowych fontów Computer Modern.

W wypadku każdego fontu wirtualnego `ecr` odpowiadający font `cmr` zachowuje się jak pojedynczy font podstawowy. Poza tym wymagany jest plik listy właściwości `pl`.

Poniżej podano kroki prowadzące do utworzenia fontu wirtualnego `ecr10` z podstawowego fontu `cmr10`. Plik instalacyjny `makeecr.tex` powinien wyglądać mniej więcej tak:

```
% This is makeecr.tex.
\input fontinst.sty
  \installfonts
    \installfamily{T1}{ecr}{}
    \installfont{ecr10}{cmr10,latin}%
        {T1}{T1}{ecr}{m}{n}{}
  \endinstallfonts
\bye
```

z tym, że potrzebne będą dodatkowe wyrażenia `\installfont` dla innych członków tej rodziny, jak kursywa, pogrubienie itd.

W celu przetworzenia TEX-em pliku instalacyjnego, należy wykonać następujące polecenia:

```
tftopl cmr10.tfm cmr10.pl
tex makeecr
vptovf ecr10.vpl ecr10.vf ecr10.tfm
rm *.log *.pl *.vpl *.mtx
```

po czym przenieść pliki `tfm` i `vf` do właściwych katalogów.

Innymi słowy, najpierw tworzymy plik listy właściwości, który zostanie wykorzystany przez TEX-a (czyli *fontinst*), wreszcie tworzymy pliki fontowe w postaci binarnej wykorzystując powstałą w trakcie przetwarzania TEX-owego listę właściwości fontu wirtualnego vpl. Na koniec usuwamy zbędne pliki (użyta została notacja unixowa). Przykład ten nie wymaga uzupełnienia psfonts.map, chyba że używa się obwiedniowej wersji fontów Computer Modern.

9.2.1. Fonty EC, dodatkowe uwagi

Podczas tworzenia ecr10.vpl *fontinst* informuje, że brakuje 34 znaków – to znaczy, że w odniesieniu do całego kompletu znaków należących do fontu kodowanego w T1 *fontinst* 34 razy nie poradził sobie ze zdefiniowaniem znaku. Wszystkie 34 znaki są bądź akcentami, bądź znakami interpunkcyjnymi, bądź znakami diakrytycznymi (literami akcentowanymi) wykorzystującymi brakujące znaki; niektórych z nich po prostu brakuje, np. symbolu funta czy francuskich cudzysłowów. Tak wygenerowany font ecr10 nie jest odpowiedni, jeśli zależy nam na dostępie do tych znaków.

Nie oszczędzamy też zbytnio na pamięci dyskowej, ponieważ plik vf wymaga około 4 kB pamięci, co jest porównywalne z wielkością pliku pk, zawierającego mapy bitowe znaków w rozdzielczości drukarki laserowej.

9.3. Instalacja fontów obwiedniowych

Przypuśćmy, że mamy do zainstalowania fonty *Adobe Garamond Roman*. Zacznijmy od zmiany nazw plików fontowych, tak by były zgodne ze standardem nazewnictwa TEX-owego. Zmieniamy nazwy plików fontowych na padr8a.pfb i padr8a.afm. Utworzymy dla przykładu font padr7t kodowany w OT1. Nowy font będzie należał do rodziny fontów pad i będzie miał desygnaty NFSS „m" i „n" (*medium series, normal shape* – szereg „medium", odmiana „zwykła"). Plik instalacyjny wygląda w tym wypadku następująco:

```
% This is file makepad.tex
\input fontinst.sty
\installfonts
 \installfamily{OT1}{pad}{}
 \installfont{padr7t}{padr8a,latin}%
    {OT1}{OT1}{pad}{m}{n}{}
\endinstallfonts
```

Prawdziwa instalacja zawiera z reguły więcej poleceń \installfont. W naszym przypadku polecenie \installfont realizuje następujące funkcje:

- konstruuje font dla rodziny pad;
- wykorzystuje informacje z padr8a.afm, uzupełniając je w razie potrzeby o instrukcje z latin.mtx;
- zadaje kodowanie OT1;
- wykorzystuje parametry OT1, pad, m i n do utworzenia pliku fd.

9.3.1. Włączanie fontów eksperckich

W wypadku większości fontów obwiedniowych jedynym sposobem otrzymania ligatur „ff", „ffi" i „ffl" jest font ekspercki, gdyż znaki te rzadko trafiają się w foncie podstawowym. Jednakże latin.mtx tworzy niepoprawne znaki z tymi nazwami, ponieważ pliki kodowania przewidują dla nich miejsce. Aby otrzymać prawidłowe ligatury z podwójnym „f" trzeba po prostu włączyć nazwę fontu eksperckiego do listy plików metrycznych polecenia \installfont. Schemat pliku instalacyjnego będzie wówczas wyglądać mniej więcej tak:

```
% This is file makepad.tex
\input fontinst.sty
\installfonts
\installfamily{OT1}{pad}{}
\installfont{padr7t}%
  {padr8a,padr8x,latin}%
  {OT1}{OT1}{pad}{m}{n}{}
\endinstallfonts
```

Proszę zwrócić uwagę na obecność padr8x, fontu eksperckiego dla fontu *Garamond.*

10. Wnioski

Powyższe przykłady demonstrują potencjalne możliwości fontów wirtualnych. Jeśli chodzi o mnie, to stosowałem fonty wirtualne do druku rozstrzelonego, do tworzenia nowych rodzin fontów matematycznych, do tworzenia fontów z podkreślanymi i przekreślanymi znakami oraz fontów *old-style*, czy do numeracji przypisów z wykorzystaniem cyfr z fontów eksperckich. Używam też technik wirtualnych aby np. dodać „j" bez kropki do moich fontów łacińskich, gdy takiego znaku brak w oryginalnym foncie podstawowym. Podsumowując chciałbym stwierdzić, że środowisko użytkowników TEX-a musi jeszcze dużo się uczyć, aby efektywnie tworzyć i wykorzystywać fonty wirtualne. Mam nadzieję, że mój krótki artykuł zachęci czytelnika do pójścia tą drogą.

◇ Alan Hoenig
ajhjj@cunyvm.cuny.edu

(*tłumaczyła Julita Bolland*)