Specjalne instrukcje specjalne

Bogusław Jackowski
BOP s.c., Piastowska 70, 80-363 Gdańsk
B.Jackowski@gust.org.pl

Krzysztof Leszczyński
Centrum Astronomiczne PAN, ul. Bartycka 18, 00-716 Warszawa
chris@camk.edu.pl

Streszczenie

W pracy zaproponowane zostało wykorzystanie specjalnego fontu jako rozszerzenia – w pewnym sensie – możliwości instrukcji `special`. Przedstawione przykłady zastosowań pokazują, że zaproponowana technika może się okazać szczególnie przydatna w przypadku METAPOST-a.

Wprowadzenie

Instrukcje specjalne (`special`) to potężne narzędzie w rękach użytkownika TEX-a, METAFONT-a i METAPOST-a. Jednakże METAPOST narzuca na instrukcje `special` poważne ograniczenie: ich zawartość jest umieszczana *na początku* generowanego pliku `EPS`, bezpośrednio po nagłówku, przed pierwszą instrukcją postscriptową wygenerowaną przez METAPOST. Oznacza to, że – inaczej niż w przypadku TEX-a czy METAFONT-a – użytkownik nie jest w stanie „przeplatać” instrukcji `special` z instrukcjami rysowania (`draw`, `fill`) czy składania tekstu (`infont`, `btex ... etex`). Utrudnia to znacznie wykorzystanie METAPOST-owych instrukcji `special` i prowadzi częstokroć do karkołomnych rozwiązań. Co gorsza, TEX-owe instrukcje `\special` są przez współpracujący z METAPOST-em program `dvitomp` ignorowane, skutkiem czego cała gama pakietów TEX-owych nie daje się wykorzystać w zastosowaniach METAPOST-owych.

Ten przykry stan rzeczy nasunął jednemu z autorów (KL) pomysł, żeby zamiast instrukcji `special` użyć specjalnego fontu; treść napisów składanych tym fontem miałaby być traktowana przez programy przetwarzające pliki `DVI`, `GF`, i METAPOST-owe `EPS`-y jak instrukcje `special`. Niniejszą pracę należy traktować jako wynik wstępnych przemyśleń – pomysł jest wciąż zbyt świeży i daleko nam jeszcze do zapanowania nad dającymi się pomyśleć konsekwencjami takiego podejścia. W szczególności nie udało się nam jeszcze stworzyć jego teoretycznych ram. Dlatego wyniki rozważań zaprezentowaliśmy na dużej liczbie przykładów, ilustrujących rozmaite sposoby wykorzystania „specjalnych instrukcji specjalnych”.

Skupiliśmy się na zastosowaniach związanych z METAPOST-em – wykorzystanie proponowanej techniki w programach TEX-owych i METAFONT-owych to osobne zagadnienie. Ocenę możliwości, jakie kryją w sobie „specjalne instrukcje specjalne”, zostawiamy Czytelnikowi. Nam się wydaje, że są one spore.

Font specjalny – kto zacz?

Font specjalny – ochrzcijmy go na wstępie imieniem `cmdfont` – zdefiniowany jest za pomocą króciutkiego programu METAPOST-owego o nazwie `cmdfont.mp`:

```
designsize:=10bp/pt - epsilon;
fontdimen 2: designsize; % font normal space
fontmaking:=1;
for i:=0 upto 255:
 beginfig(i-256);
  charwd:=charht:=chardp:=charic:=0;
 endfig;
endfor
end.
```

Przetworzenie METAPOST-em tego programu spowoduje wygenerowanie pliku metrycznego o nazwie `cmdfont.tfm`, który należy umieścić tam, gdzie przebywają wszystkie pliki `TFM` w danym systemie. Font ten zawiera 256 znaków o zerowych rozmiarach. Wszystkie parametry fontu mają także wartość zero z wyjątkiem, rzecz jasna, nominalnego rozmiaru fontu (`designsize`) i, niekoniecznie rzecz jasna, spacji (`fontdimen 2`), której nadana została wartość nominalnego rozmiaru fontu. Konkretna wartość nie jest w tym wypadku istotna, ważne jest jedynie, by spacja nie miała wielkości zerowej – w niektórych przykładach założenie takie jest niezbędne. Podobnie konkretna wartość rozmiaru nominalnego nie odgrywa roli, natomiast jest istotne, by wszyscy, którzy korzystają z tego fontu, używali *tej samej* wartości parametru `designsize`.

Użycie ujemnej wartości jako parametru makra `beginfig` (wiersz 7) powoduje, że każdy generowany EPS ma nazwę `cmdfont.ps`. Użycie pozornie naturalniejszej postaci `beginfig(i)` spowodowałoby wygenerowanie plików `cmdfont.0`, `cmdfont.1`, …, `cmdfont.255`, co zaśmieciłoby nam kartotekę roboczą, gdyż EPS-y nie są nam do niczego potrzebne i należy je po prostu wyrzucić.

Font specjalny a METAPOST

W jaki sposób font specjalny „przechodzi" do pliku METAPOST-owego? Rozważmy następujący program o nazwie, powiedzmy, `infont.mp`:

```
beginfig(100);
 draw "META FONT" infont "cmdfont";
 draw "META POST" infont "cmdfont" scaled 2;
endfig;
end.
```

Wynikiem działania METAPOST-a będzie plik `infont.100` o następującej zawartości:

```
 1. %!PS
 2. %%BoundingBox: 0 0 0 0
 3. %%Creator: MetaPost
 4. %%CreationDate: 2001.04.13:1950
 5. %%Pages: 1
 6. %*Font: cmdfont 10 10 20:80000000460708
 7. %*Font: cmdfont 20 10 20:80000000440598
 8. %%EndProlog
 9. %%Page: 1 1
10. 0 0 moveto
11. (META FONT) cmdfont 10 fshow
12. 0 0 moveto
13. (META POST) cmdfont 20 fshow
14. showpage
15. %%EOF
```

Nie wchodząc zbyt głęboko w szczegóły kodu generowanego przez METAPOST odnotujmy, że informacja o foncie `cmdfont` pojawia się w nagłówku pliku EPS (wiersze 6 i 7) oraz że napisy METAPOST-owe zostały zamienione na *pojedyncze* napisy postscriptowe (wiersze 11 i 13). Oczywiście spacja wewnątrz napisu zarówno METAPOST-owego jak i postscriptowego odnosi się do *znaku o kodzie 32* fontu `cmdfont`. (W fontach rodziny Computer Modern pod kodem 32 znajduje się znak *suppress* – przekreślnik do uzyskiwania liter „Ł" i „ł"; w większości fontów tekstowych pod kodem 32 figuruje znak *space*, dający w składzie taki sam efekt jak nierozciągliwa spacja.)

A teraz rozważmy program METAPOST-owy `btexetex.mp` składający tekst, jak sama nazwa wskazuje, z użyciem konstrukcji `btex ... etex`:

```
verbatimtex \font\f cmdfont etex
beginfig(100);
 draw btex \f META FONT etex;
 draw btex \f META POST etex scaled 2;
endfig;
end.
```

Wynikowy EPS nieco się różni od poprzedniego, co nie powinno dziwić:

```
 1. %!PS
 2. %%BoundingBox: 0 0 0 0
 3. %%Creator: MetaPost
 4. %%CreationDate: 2001.04.13:1950
 5. %%Pages: 1
 6. %*Font: cmdfont 10 10 41:8c0e1
 7. %*Font: cmdfont 20 10 41:880b3
 8. %%EndProlog
 9. %%Page: 1 1
10. 0 0 moveto
11. (META) cmdfont 10 fshow
12. 9.9999 0 moveto
13. (FONT) cmdfont 10 fshow
14. 0 0 moveto
15. (META) cmdfont 20 fshow
16. 19.99979 0 moveto
17. (POST) cmdfont 20 fshow
18. showpage
19. %%EOF
```

Istotna różnica polega na tym, że tym razem każdy z napisów TeX-owych został rozbity na dwa kawałki (wiersze 13 i 15 oraz 17 i 19), gdyż zamiast znaku o kodzie 32 TeX złożył spację, która w pliku postscriptowym została zamieniona na operację pozycjonowania. Czarnoksiężnicy, którzy potrafią czytać magiczne ciągi cyfr szesnastkowych pojawiające się w wierszach 6 i 7, bez trudu zauważą, że istotnie tym razem znak o kodzie 32 nie występuje wśród znaków wykorzystanych w składzie.

Problem spacji w foncie `cmdfont` jeszcze parokrotnie będzie się przewijał w dalszej części artykułu.

Font specjalny a TeX i `dvips`

Pliki generowane za pomocą METAPOST-a na ogół są włączane do dokumentów TeX-owych, by za sprawą sterownika `dvips` ostatecznie wylądować w pliku postscriptowym. O ile TeX-owi wystarczy plik metryczny (`TFM`), o tyle `dvips` potrzebuje dodatkowo „prawdziwego" fontu, to znaczy takiego, w którym są zapisane kształty znaków. O potrzebnych fontach `dvips` dowiaduje się z nagłówka, i jeśli nie oszukamy go zmieniając nagłówek, będzie się domagał stosownego fontu bitmapowego (`PK`) lub postscriptowego (`Type 1`).

Sztuczka, która od razu przychodzi na myśl, żeby użyć fontu wirtualnego, w którym każdy znak będzie pusty, nic nie da, gdyż `dvips` jest za sprytny

– zwęszy pułapkę i po prostu zignoruje wszystkie teksty składane takim fontem.

Nie ma innej rady, jak zgodzić się na oszukanie `dvipsa` i przez wpisanie w pliku `psfonst.map` stosownej informacji utożsamić `cmdfont` z jakimś fontem, o którym wolno zakładać, że jest zawsze obecny. Takim fontem jest na przykład Courier. Odpowiednia informacja wyglądałaby następująco:

```
cmdfont Courier
```

I to wszystko. Dzięki temu nawet pliki zawierające teksty złożone fontem `cmdfont` dadzą się oglądać, chociaż efekty mogą być zaskakujące. Jednakże font `cmdfont` powinien być tak używany, by nigdy nie doszło do próby wyświetlania jego znaków. Jak to robić? O tym traktuje kolejny punkt.

Wykorzystanie fontu specjalnego bez przetwarzania dodatkowego

Instrukcje, które trzeba okiełznać, to `cmdfont` (METAPOST do składania tekstów używa instrukcji postscriptowych o takich samych nazwach, jakie miały odpowiednie pliki TFM) i `fshow` (definicja tej operacji znajduje się w prologu `finclude.pro`, dołączanym automatycznie przez `dvipsa` po natrafieniu na EPS-y METAPOST-owe zawierające złożone teksty). Poniższy kawałek kodu METAPOST-owego chytrze redefiniuje znaczenie obu instrukcji w taki sposób, że `cmdfont` po prostu interpretuje napis spoczywający na stosie jako ciąg instrukcji (operacje `cvx exec`), po czym w sposób z lekka zawiły neutralizuje działanie oczekującej na wykonanie instrukcji `fshow`:

```
def prep_cmdfont =
 special "/fshow where ";
 special " {pop} {/fshow {pop} def} ifelse";
 special "/cmdfont {cvx exec";
 special "  /fshow.tmp /fshow load def";
 special "  /fshow";
 special "    {pop /fshow /fshow.tmp load def}";
 special "   def";
 special "} def";
enddef;
extra_endfig:=extra_endfig & ";prep_cmdfont;";
```

METAPOST-owe instrukcje specjalne, wykorzystane w powyższym kodzie, gwarantują, że redefinicje zostaną umieszczone na początku wynikowego EPS-a, co w tym wypadku jest nam na rękę.

Zauważmy, że – inaczej niż przy przetwarzaniu METAPOST-owych EPS-ów za pomocą programów zewnętrznych (por. pkt następny) – pokawałkowanie napisów w tym wypadku nie stanowi żadnego utrudnienia.

Poniżej prezentujemy trzy przykłady zastosowania takiej techniki wykorzystania fontu `cmdfont`. Efekty, jakie można za jej pomocą uzyskać, są niewątpliwie przydatnie i stosunkowo trudne do uzyskania w inny sposób. We wszystkich przykładach zakładamy, że plik `prepcmdf.mp` zawiera zdefiniowane wyżej makro `prep_cmdfont` i ustawienie zmiennej `extra_endfig`.

Przykład 1: *Kolorowanie wybranych fragmentów tekstu.* Załóżmy, że ilustracja METAPOST-owa zawiera tekst, którego fragment ma być pokolorowany:

To jest niezbyt mądry
przykład ilustrujący
możliwość pokolorowania
wybranego fragmentu
tekstu składanego
TEX-em.

Źródło METAPOST-owe, które posłużyło do wygenerowania powyższej ilustracji, jest niezbyt skomplikowane:

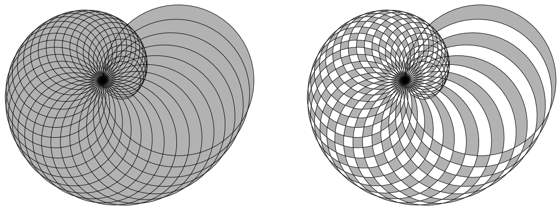
```
input prepcmdf.mp;
verbatimtex
  \def\incmyk#1#2{%
   \leavevmode\rlap{\font\f=cmdfont \f
     gsave #1 setcmykcolor}%
   #2%
  {\font\f=cmdfont \f grestore}}
etex
beginfig(100);
 draw btex \vbox{
  \hsize 40mm \pretolerance10000
   \raggedright \noindent
   To jest niezbyt mądry przykład
   ilustrujący możliwość pokolorowania
   \incmyk{0 0 0 0.3}
   {{\bf wybranego fragmentu tekstu}}
   składanego \TeX-em.
 } etex scaled 1.2;
endfig;
```

Głównym „sprawcą” pokolorowania jest dwuparametrowe makro `\incmyk`, zdefiniowane w bloku `verbatimtex ... etex`. Polecenie `\rlap`, użyte w części makra ustawiającej żądany kolor (model CMYK), jest niezbędne, gdyż nie chcemy, by tekst złożony fontem `cmdfont` wpływał na resztę składu – pamiętajmy, że spacja w tym foncie ma niezerową szerokość. A oto fragment wynikowego EPS-a, który realizuje zmianę koloru (dla poprawienia czytelności podkreślone zostały teksty zamieniane przez instrukcję `cmdfont` na operacje postscriptowe):

```
...
(gsave) cmdfont 11.99997 fshow
3.98448 28.69228 moveto
(0) cmdfont 11.99997 fshow
7.9692 28.69228 moveto
(0) cmdfont 11.99997 fshow
11.95377 28.69228 moveto
```

```
(0) cmdfont 11.99997 fshow
15.93835 28.69228 moveto
(0.3) cmdfont 11.99997 fshow
19.92294 28.69228 moveto
(setcmykcolor) cmdfont 11.99997 fshow
0 28.69228 moveto
(wybranego) plbx10 11.95514 fshow
68.84094 28.69228 moveto
(fragmen) plbx10 11.95514 fshow
117.27559 28.69228 moveto
(tu) plbx10 11.95514 fshow
0 14.3462 moveto
(tekstu) plbx10 11.95514 fshow
37.31151 14.3462 moveto
(grestore) cmdfont 11.99997 fshow
...
```

Przykład 2: *Implementacja operatora* `eofill`. PostScript został wyposażony w dwie podstawowe operacje wypełniania konturu: `fill` i `eofill` (*even-odd fill*). Różnicę między tymi dwoma sposobami wypełnienia ilustruje poniższy rysunek – „ślimaczki” zostały skonstruowane z kółek wypełnionych za pomocą operacji `fill` (lewa strona) i `eofill` (prawa strona):



Niestety METAPOST korzysta jedynie z operatora `fill`. Operator `eofill` daje się zaimplementować za pomocą METAPOST-owych instrukcji specjalnych, jednakże w ogólnym wypadku jest to niezwykle kłopotliwe. Inny sposób, równie niewygodny, to przetwarzanie zewnętrzne.

Użycie fontu `cmdfont` pozwala na względnie prostą implementację operacji `eofill`:

```
1. def eofill(text paths) text modif =
2.  begingroup
3.   save x_, y_;
4.   for p_:=paths:
5.    x_:=xpart(llcorner(p_));
6.    y_:=ypart(llcorner(p_));
7.    exitif true;
8.   endfor
9.   draw ("/fill.tmp /fill load def " &
10.    "/newpath.tmp /newpath load def" &
11.    "/fill {/fill{}def /newpath{}def}def")
12.    infont "cmdfont" shifted (x_,y_) modif;
13.   for p_:=paths: fill p_ modif; endfor
14.   draw ("eofill /fill /fill.tmp load def " &
15.    "/newpath /newpath.tmp load def")
16.    infont "cmdfont"  shifted (x_,y_) modif;
17.  endgroup
18. enddef;
```

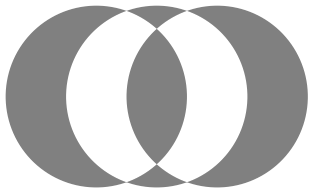
Parę słów komentarza: wiersze 9 – 12 neutralizują operacje `fill` i `newpath`, które pojawią się w kodzie postscriptowym powstałym z wiersza 13; wiersze 14 – 16 wywołują operację `eofill` i odtwarzają znaczenie operacje `fill` i `newpath`. Pozycjonowanie napisów składanych fontem `cmdfont` wymaga zadbania, żeby nie uległy zmianie rozmiary obrazka. Napisy są pozycjonowane w lewym dolnym rogu pierwszej z wypełnianych krzywych (wiersze 4 – 8) – to wystarczy, bo wszystkie napisy mają szerokość 0 (tym razem spacje są interpretowane jako znaki o kodzie 32).

Przykładowe wywołanie makra `eofill` mogłoby wyglądać następująco:

```
eofill(
 fullcircle scaled 24mm shifted (-8mm,0),
 fullcircle scaled 24mm,
 fullcircle scaled 24mm shifted (8mm,0))
 withcolor 1/2white;
```

Tak zaimplementowana operacja `eofill` od operacji `fill` różni się z punktu widzenia użytkownika tym, że argumentem zamiast pojedynczej ścieżki jest lista ścieżek.

Wynik działania powyższego kodu nietrudno przewidzieć:



Wynikowego kodu postscriptowego nie podajemy, żeby Czytelnika nie zanudzić na śmierć, ale jednak zachęcamy do postudiowania kodu przy nadarzającej się okazji – jest to bardzo pouczająca lektura.

Przykład 3: *Implementacja operatora* `eoclip`. Jeśli chodzi o operację przycinania obrazków (*clipping*), to, podobnie jak w przypadku wypełniania krzywych, METAPOST oferuje tylko operator `clip`, podczas gdy PostScript – `clip` i `eoclip`.

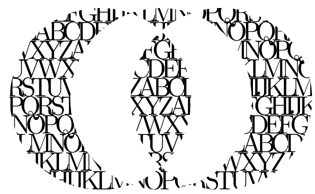
Implementacja operatora `eoclip` korzystająca z fontu `cmdfont` stanowi najtrudniejszy z omawianych w tym punkcie przykładów. A oto nasza propozycja implementacji:

```
def eoclip(expr pic)(text paths) text modif =
 begingroup
  save s_, xmin_, xmax_, ymin, ymax_;
  xmin_=ymin_:=infinity; xmax_=ymax_:=-infinity;
  draw ("/clip.tmp /clip load def " &
   "/newpath.tmp /newpath load def" &
   "/clip {/clip{}def /newpath{}def}def")
   infont "cmdfont";
  picture s_;
  s_:=image(
```

```
11.    draw ("eoclip /clip /clip.tmp load def " &
12.    "/newpath /newpath.tmp load def")
13.    infont "cmdfont"; draw pic);
14.   for p_:=paths: clip s_ to p_ modif;
15.    if xpart(llcorner(p_ modif)) < xmin_:
16.     xmin_:=xpart(llcorner(p_ modif)); fi
17.    if xpart(urcorner(p_ modif)) > xmax_:
18.     xmax_:=xpart(urcorner(p_ modif)); fi
19.    if ypart(llcorner(p_ modif)) < ymin_:
20.     ymin_:=ypart(llcorner(p_ modif)); fi
21.    if ypart(urcorner(p_ modif)) > ymax_:
22.     ymax_:=ypart(urcorner(p_ modif)); fi
23.   endfor
24.   setbounds s_ to
25.     (xmin_,ymin_)--(xmax_,ymin_)--
26.     (xmax_,ymax_)--(xmin_,ymax_)--cycle;
27.   addto currentpicture also s_;
28.  endgroup
29. enddef;
```

Zaproponowane przez nas rozwiązanie nie jest bynajmniej oczywiste, ma swoje wady, i niewykluczone, że w przyszłości wspólnymi siłami dopracujemy się lepszego rozwiązania. Nie wdając się więc w zbyt drobiazgową analizę zaproponowanego kodu zwróćmy uwagę na parę istotnych punktów: (1) część tekstu składanego fontem `cmdfont` jest dopisywana bezpośrednio do bieżącego obrazka (`currentpicture`; wiersze 5–8), a część do lokalnego obrazka `s_` (wiersze10–13) – chodzi tu zapanowanie nad kolejnością operacji w wynikowym pliku `EPS`; (2) tekst tym razem pozycjonowany jest w początku układu współrzędnych, gdyż i tak koniec końców obrazek otrzymuje odpowiednie rozmiary (wiersze 24–26); (3) sposób użycia bardziej przypomina zdefiniowany w poprzednim przykładzie `eofill` niż macierzysty `clip` – można, i chyba trzeba, się do tego przyzwyczaić…

Poniższa ilustracja przedstawia efekt użycia instrukcji `eoclip`:



Do jej wygenerowania posłużył następujący, niezbyt wymyślny program:

```
beginfig(100);
 picture p; p:=btex \vbox{\hsize 45mm
  \spaceskip-2ptplus1pt \parfillskip0pt
  \baselineskip7pt \lineskiplimit-\maxdimen
  \noindent
   A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
   A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
   A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
   A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
   A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
   A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
   A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
   A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
   A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
   A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
 } etex;
 eoclip(p)(
  fullcircle scaled 24mm shifted (-8mm,0),
  fullcircle scaled 24mm,
  fullcircle scaled 24mm shifted (8mm,0))
  shifted center p;
endfig;
```

Mamy nadzieję, że zaprezentowane przykłady przekonają Czytelnika, że używanie takich operacji jak `eofill` i `eoclip` jest proste, co oczywiście nie oznacza, że ich definicje są proste. Wydaje się nam jednak, że tworzenie takich definicji leży w zasięgu możliwości każdego użytkownika METAPOST-a – liczymy na to, że miłośnicy METAPOST-a wymyślą całą stertę innych użytecznych operacji, które za pomocą techniki opisanej w tym punkcie można definiować.

Font specjalny a pakiety TEX-owe

Jednym z zastosowań fontów specjalnych może być przetwarzanie fragmentów TEX-a na poziomie METAPOST-a. Podstawowa konstrukcja METAPOST-owa, czyli `btex ...` kod TEX-owy `... etex`, działa poprawnie pod warunkiem używania wyłącznie *klasycznego* TEX-a. Dlatego fraza:

```
picture P;
P:=btex \epsfbox{plik.eps} etex;
```

nie da oczekiwanych wyników. Powodem jest obecność TEX-owych instrukcji `\special`, które, jak to zostało odnotowane na wstępie, zostaną zignorowane w trakcie generowania pliku `EPS`. Makro `\epsfbox`, zdefiniowane w pliku `epsf.tex` lub `epsfx.tex`, analizuje plik postscriptowy i generuje odpowiedni „wihajster” (*whatsit*) typu `\special`. Na przykład plik TEX-owy

```
\input epsf \epsfbox{tiger.ps} \end
```

gdzie `tiger.ps` jest popularnym, „dyżurnym” plikiem `EPS` pochodzącym z dystrybucji `Ghostscripta`, produkuje `\special{PSfile=tiger.ps llx=22 lly=171 urx=567 ury=738 rwi=5450}`. W przypadku TEX-a włączanego do METAPOST-a powinniśmy instrukcję `\input epsf` zawrzeć wewnątrz bloku `verbatimtex ... etex`.

Jak widać, bez instrukcji `\special` – ani rusz! Trzeba zatem znaleźć sposób na przekazanie METAPOST-owi informacji o poleceniach `\special` zawartych w pliku `DVI`. Jednym ze sposobów jest takie przedefiniowanie polecenia `\special`, żeby skutkiem jego działania było złożenie fontem `cmdfont` danego ciągu znaków. Rozwiązanie naiwne:

```
\def \special #1{\smash{\hbox{%
        \cmdfont TeXspecial #1}}}
```

niestety nie będzie zupełnie dobrze działać, gdyż argument instrukcji `\special` na ogół zawiera spacje. Spacje (o niezerowej wielkości) TEX w składzie zastąpi klejami, a to z kolei spowoduje rozbicie argumentu na kilka napisów w wynikowym kodzie postscriptowym (po przetworzeniu sterownikiem `dvips`), pomiędzy którymi wystąpią instrukcje przesunięcia bieżącego punktu pozycjonowania napisów. W dodatku `\special` może zawierać znaki o zupełnie niespodziewanych kategoriach, na przykład `$`. Załóżmy zatem, że instrukcja `\special` otrzymuje jako argument ciąg znaków o różnych być może kategoriach, ale bez nierozwijalnych poleceń, takich jak `\advance`, które teoretycznie mogłyby się pojawić (ale po co?). Musimy przekształcić cały napis na inny, zawierający ten sam ciąg znaków, tyle że o kategoriach „drukowalnych".

Możemy się posłużyć klasycznym trickiem: konstrukcja `\csname ... \endcsname` pozwoli nam na całkowite rozwinięcie zawartości argumentu. Przy okazji dostaniemy informację o możliwych błędach, gdyż `\csname` „wyłoży" się na poleceniach, które nie rozwijają się do znaków. W ten sposób zostanie skonstruowana sekwencja sterująca o tymczasowej wartości `\relax`, która może być użyta jako argument do rozwijalnego polecenia `\string`. Polecenie to zamienia sekwencję sterującą na odpowiedni ciąg znaków o kategorii 12 (znaki „inne"), więc po usunięciu początkowego znaku \ dostaniemy to, co chcieliśmy mieć... niestety nie! Nie wszystkie znaki otrzymają kategorię 12 – spacjom zostanie przypisana „tradycyjna" kategoria 10. Dlatego musimy jeszcze rekurencyjnie zamienić wszystkie spacje na jawne polecenie `\char32`, dzięki czemu TEX zamiast odstępu zapisze do pliku `dvi` znak o kodzie 32 z fontu `cmdfont`, co `dvips` z kolei zamieni na zwykłą spację. Uproszczony (acz działający) plik `cmdfont.tex`, przedefiniowujący znaczenie instrukcji `\special`, może wyglądać tak:

```
\font \cmdfont=cmdfont
% pudełko do budowania łańcucha znaków, który
% będzie użyty zamiast instrukcji special.
\newbox \mpspecialbox

% do MVLspecialbox zbierzemy polecenia special, które
% by weszły do MVL (main vertical list).
\newbox \MVLspecialbox
\setbox \MVLspecialbox=\null
\def \mpspecial #1{%
   \setbox \mpspecialbox=\hbox{%
       \cmdfont
       % na wszelki wypadek ustawmy escapechar.
       \escapechar=`\\%
% zbudujmy makro o nazwie będącej treścią naszego
% speciala, wraz ze znaczkiem \ na początku
       \edef \a {\expandafter \string
          \csname TeXspecial: #1\endcsname
          \space \relax}%
       % \b zje nam „\" które powstało na początku
       % \a, po rozwinięciu
       \def \b ##1{\c}
       % Zamieniamy spacje na znak o kodzie 32
       \def \c ##1 ##2\relax{##1%
          \ifx $##2$%
          \else \char32 \c##2\relax
          \fi}%
       \expandafter \b \a
   }%

   % Jeśli jesteśmy w trybie pionowym głównym
   % to należy zachować special w specjalnym pudełku
   \if \ifvmode\ifinner+\else-\fi\else+\fi +%
       \box \mpspecialbox
   \else
       \global \setbox \MVLspecialbox
            \hbox{\box \MVLspecialbox
                   \kern1sp
                   \box \mpspecialbox}%
   \fi
   }

\def\special{\mpspecial}
```

W ten sposób mamy zagwarantowane, że METAPOST zamiast instrukcji `\special` otrzyma do złożenia fontem `cmdfont` ciąg znaków. Co ważniejsze, w pliku postscriptowym każdy taki ciąg będzie reprezentowany jako pojedynczy napis. Słowa komentarza wymaga końcowy fragment, zawarty w instrukcji warunkowej `\ifinner` – wrócimy do tego tematu później.

Problem w tym, co z tymi napisami, które pojawią się w wygenerowanym przez METAPOST-a pliku EPS, dalej począć? Tym razem nam się nie upiecze – będziemy musieli przetworzyć otrzymany plik albo ręcznie, albo automatycznie za pomocą takich programów jak `sed`, `awk` czy `perl`. Dzięki temu, że są to pojedyncze napisy, przetwarzanie nie będzie horrendalnie trudne.

Przetwarzanie pliku postscriptowego

Zróbmy mały przykład. Utwórzmy plik `tygrys.mp`:

```
verbatimtex
  \input cmdfont
  \input epsf
etex

beginfig(100)
  draw btex \epsfxsize=20pt
        \epsfbox{tiger.ps} etex
endfig;
end.
```

Po przetworzeniu otrzymamy plik `tygrys.100` o zawartości:

```
%!PS
%%BoundingBox: 0 0 20 21
%%Creator: MetaPost
%%CreationDate: 2001.04.04:1968
%%Pages: 1
%*Font: cmdfont 10 10 20:800277e4000098805748bdc
%%EndProlog
%%Page: 1 1
9.9626 0 moveto
(TeXspecial: PSfile=tiger.ps llx=22 lly=171 u\
rx=567 ury=738 rwi=199) cmdfont
 10 fshow
showpage
%%EOF
```

Parametry `llx`, `lly`, `urx`, `ury` określają rozmiary rysunku (*bounding box*), zaś `rwi` jest równe `\epsfxsize` $\times$ 10/`1bp`. W takim razie musimy przetworzyć plik wynikowy w taki sposób, by napisy złożone fontem `cmdfont` zastąpić kodem, jaki wygenerowałby `dvips`, gdyby w pliku `DVI` natrafił na instrukcję `\special` tejże treści. Musimy więc uważać na wiersze zaczynające się od `(TeXspecial:` i przetwarzać je aż do końcowego `) cmdfont` i `fshow`.

Co rozumie `dvips`

Sterownik `dvips` rozumie tylko niektóre instrukcje `\special`. „Swoje” od „obcych” odróżnia po prefiksie, czyli ciągu znaków rozpoczynających argument. Poniżej lista najważniejszych prefiksów, rozpoznawanych przez `dvipsa`:

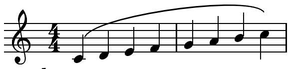
- `papersize` – definiuje rozmiar strony; wydaje się, że najlepszym rozwiązaniem jest pominięcie tej postaci w trakcie przetwarzania plików `EPS`.
- `landscape` – specyfikuje położenie strony; podobnie jak `papersize` kwalifikuje się do pominięcia.
- `header` – dołącza plik (o nazwie wyspecyfikowanej w dalszej części argumentu) do nagłówka pliku postscriptowego. Ponieważ pliki wygenerowane METAPOST-em są na ogół włączane do plików TEX-owych, tę konstrukcję można „sklonować” i dołączyć z powrotem do pliku TEX-owego jako „zwykły” `\special`. Gdyby program dvips wzruszał się postscriptowym komentarzem strukturalnym `%%DocumentNeeded-Resources`, to można by wprost każdy `header` zamienić na taki komentarz. Problem w tym, że `dvipsa` takie coś nie wzrusza, a zatem musi nas wzruszyć osobiście, o czym za chwilę.
- `psfile` – dołącza plik `eps`. Niestety, prawdopodobnie nie ma innej rady jak tylko zrobić to „ręcznie”, a ściślej skryptowo. Pewne elementy, takie jak deklarację `%*Font` można sklonować do pliku pomocniczego.
- `!` – kawałek kodu postscriptowego, który powinien być przeniesiony do nagłówka. Można go sklonować podobnie, jak `header`.
- `"` – kawałek kodu postscriptowego, umieszczany w takim środowisku (układzie współrzędnych), jakie obowiązuje w plikach `EPS`.
- `ps:` – kawałek kodu postscriptowego, umieszczany w takim środowisku (układzie współrzędnych), jakie obowiązuje w pliku `PS` wygenerowanym przez `dvipsa`; w kodzie postscriptowym przed treścią argumentu i po niej `dvips` umieszcza operacje pozycjonowania.
- `ps::`, `ps::[begin]`, `ps::[end]` – te konstrukcje służą do łączenia występujących bezpośrednio po sobie instrukcji `\special`; wróżą kłopoty – ich opis w dokumentacji `dvipsa` jest bardzo skąpy, toteż używając ich można nieźle sobie naPSocić – dokumentacja niedwuznacznie przestrzega przed bezrozumnym używaniem tych instrukcji: *you should understand what you're doing*; ponadto przydatność tego rodzaju zabiegów jest niejasna, załóżmy więc, że w plikach, które będziemy przetwarzać, takie instrukcje nie wystąpią.
- `em:` – instrukcje `\special` wymyślone przez Eberharda Mattesa, zrozumiałe ongiś tylko dla sterowników pakietu `emTeX`; być może wygodne, ale tylko wówczas, gdy się nie korzysta z dobrodziejstw PostScriptu. Podobnie jak poprzednio założymy, że nie będziemy się musieli z nimi borykać.
- `html:` – to może innym razem `:-)`

Klonowanie konstrukcji

Zakładając, że każdy plik METAPOST-owy w końcu przejdzie jeszcze raz przez TEX-a oraz przez `dvipsa`, można uniknąć części pracy „klonując” pewne konstrukcje. Instrukcję `\special` w postaci `header` oraz „`!`” (zaczynającą się od wykrzyknika), można przepisać do tymczasowego pliku TEX-owego, który następnie wczytamy. Podobnie komentarze strukturalne typu `%*Font:` w pliku, który jest wczytywany za pomocą instrukcji `\special{psfile ... }`, można przenieść tego samego tymczasowego pliku TEX-owego.

Do przetwarzania plików wygenerowanych za pomocą METAPOST-a służy program `despecials` (dostępny w archiwum BachoTEX 2001). Jest to skrypt perlowy, który *unaczeŚnia* pliki wynikowe METAPOST-a w ten sposób, że zamienia opisane wyżej konstrukcje na odpowiednie (bądź czasami nieodpowiednie!) konstrukcje postscriptowe. Produkuje też plik TeX-owy `mpspec.inc` zawierający sklonowane polecenia `\special` o charakterze globalnym.

Pliki nagłówkowe

Jest gorzej, jeśli plik TeX-owy, który dołączamy, generuje polecenia `\special` w plikach bibliotecznych. Tak robi wiele zbiorów makr, na przykład `pstricks` albo `lilypond`. Dla ilustracji weźmy  jeden z bardzo prostych przykładów, wygenerowanych programem `lilypond`. Efektem pracy `lilyponda` jest plik TeX-owy, który można włączyć do programu METAPOST-owego. Autorzy `lilyponda` stwierdzili wyraźnie, że pliki, produkowane przez ten program, są *special-infested*. Rzeczywiście, nawet tak prosty przykład zawiera ponad cztery special-e: dwa generujące nagłówki oraz dwa generujące łuczek. W tak skomplikowanej materii, jaką jest skład nutowy, nie uniknie się stosowania dużej ilości wstawek postscriptowych. W małym fragmencie menueta, zamieszczonego w biuletynie numer 14, jest ponad 60 poleceń `\special`. Nasz plik METAPOST-owy mógłby wyglądać tak:

```
verbatimtex
 \input cmdfont
 \input lilyponddefs
etex
beginfig(100)
 picture P;
 P=btex \input gama-c.tex etex;
 draw P;
endfig;
end.
```

Niestety, makra zawarte w pliku `lilyponddefs.tex` same generują polecenia `\special`. Spójrzmy, jaki będzie efekt. METAPOST generuje plik `mpx$$.tex` (na różnych systemach nazwy mogą się różnić). Zawartość pliku jest niemal oczywista:

```
\input cmdfont
\input lilyponddefs
%
\shipout\hbox{\smash{\hbox{\hbox{%
\input gama-c.tex}\vrule width1sp}}}
\end{document}
```

Widzimy tu dwa wiersze, pochodzące z konstrukcji `verbatimtex ... etex`, oraz zawartość naszego bloku TeX-owego, zanurzoną w dość skomplikowanej składance pudełek, które są wysyłane jawnie do pliku .dvi poleceniem `\shipout`.

Bez specjalnych zabiegów efektem TeX-owania takiego pliku będzie plik DVI zawierający *dwie* strony. Pierwszą stronę wygeneruje nam `\shipout` i będzie ona zawierała treść pudełka, niestety bez niezbędnych informacji o nagłówku. Drugą stronę wygeneruje instrukcja `\end` – przetworzona zostanie zawartość głównej listy pionowej TeX-a i wynikowa strona będzie zawierała deklaracje nagłówków.

Nie ma jednoznacznej odpowiedzi na pytanie, co z tym fantem zrobić. Rozwiązanie zależy od tego, co byśmy chcieli uzyskać. Jednym ze sposobów jest zebranie wszystkich poleceń `\special`, a ściślej całej listy pionowej i dokładanie jej do każdego pudełka – jawnie lub niejawnie, poprzez przedefiniowanie polecenia `\shipout`.

Załóżmy, że pliki nagłówkowe umieszczają w głównej liście pionowej wyłącznie polecenia `\special`, bez zanurzania w pudełka. Wówczas możemy łatwo odróżnić `\special` nagłówkowy od tego generowanego w `btex ... etex`. Pierwsze z nich są generowane wyłącznie na liście głównej pionowe, drugie w trybie wewnętrznym poziomym lub ograniczonym pionowym. Możemy je rozróżnić parą warunków `\ifinner` `\ifvmode`; `cmdfont.tex` „wyłapuje” instrukcje `\special`, które „miałyby ochotę” znaleźć się na głównej liście pionowej i dokłada je do specjalnego pudełka `\MVLspecialbox`. Pudełka są rozdzielane pojedynczą cienką spacją o szerokości 1sp, gdyż inaczej zawartość dwóch kolejnych instrukcji `\special` mogłaby zostać sklejona na skutek dopuszczalnej operacji `\unhbox`.

W przypadku gdy nasze makra generują wyłącznie polecenia `\special`, przechwycone przez `\mpspecial`, wówczas rejestr `\pagetotal` będzie równy 0pt. Jeśli nie jest, to znaczy, że coś się dostało na listę główną, a to prawdopodobnie oznacza błąd. Tego przypadku nie da się automatycznie obsłużyć w sposób ogólny. Jeśli jednak taka sytuacja ma miejsce, to możemy łatwo zebrać całą listę pionową, o ile nie przekroczyła wysokości strony. Możemy to zrobić w taki sposób:

```
\par
\newbox \MVLbox
\begingroup
   \ifdim \pagetotal>0pt
      \errhelp{Zignoruję MVL.}
      \errmessage{MVL niepusta}
      \output={\global\setbox
         \MVLbox=\box255}
      \vsize=\pagetotal
      \eject
   \fi
\endgroup
```

To, co zrobimy z zawartością `\MVLbox`, zależy już całkowicie od nas.

Zakończenie

W świetle przedstawionych przykładów propozycja, by programy TeX, METAFONT i METAPOST, a także programy z nimi współpracujące, traktowały teksty składane fontem specjalnym jako instrukcje specjalne, zapowiada się obiecująco, zwłaszcza jeśli chodzi o rozszerzenie możliwości METAPOST-a, a być może także TEX-a i METAFONT-a.

Ocena, na ile użycie fontu specjalnego byłoby przydatne w przypadku TEX-a i METAFONT-a, wymaga jeszcze doświadczeń. Doświadczeń wymaga także ustalenie szczegółów technicznych budowy fontu. Nie jest do końca jasne, jakie parametry ów specjalny powinien font mieć definiowane i jaka powinna być ich wartość, na przykład jaka powinna być szerokość spacji – 0, 1sp, $\frac{1}{3}$em (typowa wielkość dla fontu tekstowego), czy 1em (jak w zaprezentowanym tu przypadku, dzięki czemu można bez trudu z wnętrza programu TEX-owego dowiedzieć się, jaka jest wielkość parametru `designsize`). Nie jest też jasne, czy nie należałoby dopuścić używania rodziny fontów specjalnych, a jeżeli tak – to jakich reguł się trzymać, żeby nie dopuścić do powstania bałaganu. Bałagan bowiem, a dokładniej brak standardu, może się okazać najgroźniejszym wrogiem efektywnego wykorzystania zaproponowanej technologii. Standaryzacja, i to standaryzacja możliwie szybka, obejmująca zarówno nazwę i budowę fontu, jak i postać tekstów składanych fontem specjalnym, jest warunkiem *sine qua non* sukcesu. Mamy nadzieję, że środowisko TEX-owców będzie w stanie sprawnie zadziałać zanim powstanie bałagan, który tę dobrze zapowiadającą się technologię może rozłożyć na łopatki.

Podziękowania

Autorzy wyrażają wdzięczność Redakcji Biuletynu za okazaną cierpliwość.

Bogusław Jackowski
BOP s.c., Piastowska 70,
80-363 Gdańsk
B.Jackowski@gust.org.pl

Krzysztof Leszczyński
Centrum Astronomiczne PAN,
ul. Bartycka 18,
00-716 Warszawa
chris@camk.edu.pl