System MetaPost

John D. Hobby

Sierpień 1990

Tłumaczenie za zgodą autora
Joanna Marszałkowska

Streszczenie

MetaPost jest językiem programowania służącym do tworzenia rysunków wektorowych. Sposób działania systemu MetaPost jest bardzo podobny do działania systemu METAFONT, którego autorem jest Donald E. Knuth. Główną różnicą między jednym a drugim jest rodzaj generowanego pliku wyjściowego. METAFONT tworzy mapę bitową, natomiast MetaPost generuje plik PostScript-owy, który może być dołączony do wszystkich dokumentów drukowanych na drukarkach PostScript-owych. Ponadto MetaPost ułatwia dostęp do wszystkich funkcji PostScript-u oraz zawiera narzędzia umożliwiające łączenie tekstu z grafiką.

Dokument ten opisuje budowę i działanie języka MetaPost. Zawiera podstawową dokumentację, która może być użyta w połączeniu z podręcznikiem *The* METAFONT*book*. Wiele plików źródłowych MetaPost-a zostało zaczerpniętych wprost ze źródeł METAFONT-a (za zgodą autora Donalda E. Knuth'a).

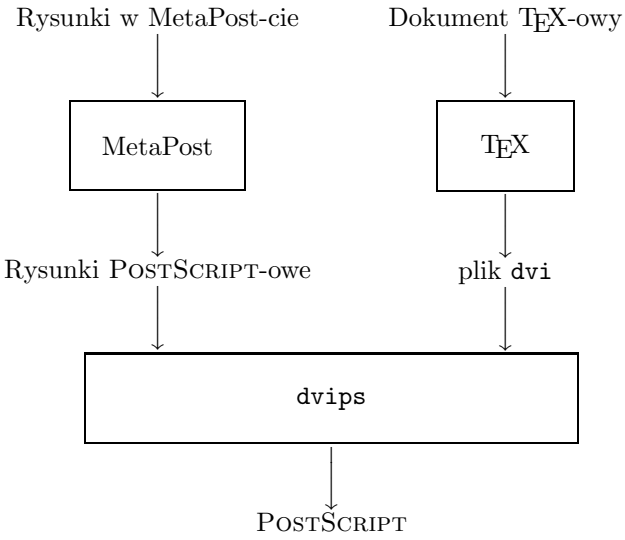
1. Wstęp

Pierwowzorem systemu MetaPost jest METAFONT[1] [3]. Wiele z plików źródłowych MetaPost-a zostało zaczerpniętych bezpośrednio ze źródeł METAFONT-a. Podobnie jak METAFONT, MetaPost jest językiem programowania służącym do tworzenia rysunków. Dodatkowo został poszerzony o nowe komendy podstawowe, służące do łączenia tekstu z grafiką, które wykorzystują możliwości PostScript[2]-u, takie jak odcinanie (ang. *clipping*), cieniowanie (ang. *shading*), czy rysowanie linii przerywanych (ang. *dashed lines*). MetaPost posiada też wszystkie podstawowe możliwości METAFONT-a, takie jak na przykład rysowanie krzywych i obrazków, przekształcenia afiniczne czy określanie kształtu piórka (ang. *pen shape*). Inną umiejętnością zapożyczoną od METAFONT-a jest zdolność rozwiązywania układów równań liniowych, które dane są w postaci niejawnej.

[1] METAFONT jest znakiem handlowym firmy Addison Wesley Publishing.

[2] PostScript jest znakiem handlowym firmy Adobe Systems Inc.

MetaPost może być też użyty jako narzędzie do tworzenia fontów PostScript-owych. Jednak jego głównym zastosowaniem jest generowanie rysunków umieszczanych później w dokumentach TEX-owych[3] oraz *troff*-owych. Rysunki są łączone w całość z dokumentem TEX-owym za pomocą powszechnie dostępnego programu `dvips` w sposób przedstawiony na Rysunku 1.[4] W przypadku *troff*-a, plik wyjściowy jest generowany przez program `dpost`, a rysunki są umieszczane w dokumencie za pomocą *troff*-owej komendy `\X`.



Rysunek 1. Diagram przetwarzania dokumentu TEX-owego z rysunkami przygotowanymi przez program MetaPost.

Poza nowymi poleceniami do łączenia tekstu i grafiki oraz poszerzeniem o możliwości jakie daje PostScript, główną różnicą między tymi językami jest fakt, że MetaPost generuje rysunki obwiedniowe (ciągłe, zdefiniowane za pomocą krzywych), natomiast METAFONT w postaci map bitowych (dyskretne). Wzajemne oddziaływanie zmiennych systemowych oraz niektóre bardziej zaawansowane aspekty języka zostały naszkicowane w następnych dwóch rozdziałach.

Części 2. i 3. tego dokumentu zawierają pobieżny opis języka wraz z ponumerowanymi przykładami. Część 4. opisuje implementację. Wstępna prezentacja języka znajduje się w [1].

2. Wprowadzenie do systemu MetaPost

MetaPost jest bardzo podobny do systemu METAFONT napisanego przez Donalda E. Knuth'a. Główną różnicą jest fakt, że zamiast bitmapy generuje plik

[3] TEX jest znakiem handlowym firmy American Mathematical Society.

[4] Plik źródłowy w C, służący do utworzenia `dvips` pochodzi z dystrybucji web2c TEX-a. Podobne programy są dostępne również z innych źródeł.

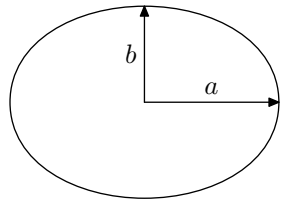
PostScript-owy. Knuth opisał dokładnie język METAFONT w podręczniku *The* METAFONT*book* [4].

Ten dokument prezentuje system MetaPost poprzez przykłady i odwoływanie się do kluczowych fragmentów podręcznika *The* METAFONT*book*. Dobrym pomysłem jest rozpoczęcie czytania od Rozdziałów 2. i 3. Jedyną różnicą między językiem METAFONT, a MetaPost występującą w tych rozdziałach jest fakt, że domyślną jednostką w MetaPost-cie jest punkt PostScript-owy (w TEX-u *bp* – big point, $72bp = 1\text{cal}$).

Aby zobaczyć MetaPost w akcji, przyjrzyjmy się plikowi `fig.mp`, który zawiera następujący kod:

```
beginfig(1);
a=.7in; b=0.5in;
z0=(0,0); z1=(a,0); z2=(0,b);
z0=.5[z1,z3]=.5[z2,z4];
draw z1..z2..z3..z4..cycle;
drawarrow z0..z1;
drawarrow z0..z2;
label.top(btex $a$ etex, .5[z0,z1]);
label.lft(btex $b$ etex, .5[z0,z2]);
endfig;
end;
```

Polecenie `mp fig.mp` powoduje wygenerowanie pliku PostScript-owego `fig.1`, który może być umieszczony w dokumencie TEX-owym. Po dołączeniu do takiego dokumentu pakieciku makr `epsf`* (`\input epsf`) i napisaniu polecenia `$$\epsfbox{fig.1}$$`, otrzymamy:



Linia `beginfig(1)` oznacza, że wszystkie polecenia aż do `endfig` służą do utworzenia rysunku `fig.1`. Jeśli w pliku `fig.mp` znajdzie się więcej rysunków, definicja każdego z nich rozpocznie się poleceniem `beginfig(`*nr*`)` i zakończy `endfig`.

W MetaPost-cie dodatkowo zostało utworzone makro `drawarrow`, które rysuje strzałkę. Zdefiniowano też polecenie `drawdblarrow`, które rysuje podaną ścieżkę z grotami strzałek na obu końcach. Linia zaczynająca się poleceniem `label.top` jest wywołaniem standardowego makra służącego do umieszczania tekstu nad podanym punktem. W tym przypadku, współrzędne punktu `.5[z0,z1]` są określone jako połowa odległości pomiędzy punktami `z0` i `z1`, a napis `$a$` spowoduje wygenerowanie literki *a* w trybie matematycznym TEX-a. Oprócz poleceń `label.top` i `label.lft` dostępne są także polecenia `label.bot`, `label.rt` oraz cztery dodatkowe wersje: `label.ulft` (w lewo i do góry), itd.

* z pakietu `dvips` Thomasa Rokickiego (przyp. tłum.).

Piórka używane przez MetaPost zostały dokładnie opisane w Rozdziale 4. *The* METAFONT*book*. Proste rysunki nie wymagają jednak dokładnego określenia piórka. Piórko domyślnie używane przez MetaPost ma kształt koła. o średnicy $0.5bp$[5], które daje linię o stałej grubości $0.5bp$, bez względu na przebieg rysowanej krzywej (patrz przykład 5. na str. 10).

Zawartość Rozdziału 5. *The* METAFONT*book* nie we wszystkim odnosi się do MetaPost-a. W szczególności, MetaPost nie generuje plików `gf` oraz nie zawiera makra `mode_setup`. MetaPost posiada zbiór predefiniowanych makr, jednak nie są one takie same jak plainowa baza (format) METAFONT-a. Warto wspomnieć, że MetaPost posiada preprocesor, który czyta zawartość bloku `btex ... etex`, następnie tłumaczy ją na polecenia niskiego poziomu, rozumiane przez MetaPost. I tak, jeśli głównym plikiem jest `fig.mp`, to przetłumaczony materiał TEX-owy zostaje umieszczony w pliku o nazwie `fig.mpx`. Zazwyczaj plik ten nie odgrywa większego znaczenia dla użytkownika. Dopiero gdy w bloku `btex ... etex` wystąpi błąd (np. błędna komenda TEX-owa), jego zawartość zostaje skopiowana do pliku `mpxerr.tex`, a informacje o błędach zostają umieszczone w pliku `mpxerr.log`.

Jeśli zaistnieje potrzeba odwołania się do polecenia TEX-owego lub wcześniej zdefiniowanego makra, można je umieścić w bloku `verbatimtex ... etex`. Różnica między blokiem rozpoczynającym się od `btex`, a `verbatimtex` jest taka, że zawartość pierwszego jest przekształcana w wyrażenie typu `picture` (np. litera w obrazek litery).

W systemie UNIX zmienne środowiskowe mogą być tak użyte, że zawartości bloków `btex ... etex` oraz `verbatimtex ... etex` mogą być napisane w *troff*-ie, a nie w TEX-u. W takiej sytuacji dobrym pomysłem jest napisanie polecenia `prologues:=1`, które podpowie MetaPost-owi jak wygenerować plik wyjściowy i spowoduje, że tekst zostanie złożony wbudowanymi fontami PostScript-owymi.

Główne elementy języka MetaPost są identyczne z opisanymi w Rozdziałach 6–10 *The* METAFONT*book*. Jedyną zmianą wprowadzoną w procesie zamiany znaków z pliku źródłowego na żetony (opisanym w Rozdziale 6) jest fakt, że materiał TEX-owy (np. tekst) może zawierać znaki procentu (`%`) oraz cudzysłów (`"`), które w trakcie tłumaczenia na MetaPost będą traktowane jako odstępy.* Warto przy tym dodać, że w pełni zachowuje zawartość bloku `btex ... etex`, z wyjątkiem spacji początkowych i końcowych.

Rozdziały 7–10 zawierają opisy typów zmiennych i wyrażeń rozumianych przez METAFONT. MetaPost posiada dodatkowy typ `color`, który jest podobny do typu `pair`, tylko zamiast dwóch posiada trzy współrzędne. Na zmiennych typu `color` można wykonywać operacje dodawania i odejmowania oraz mnożenia i dzielenia przez skalar. MetaPost rozumie także wyrażenia podane niejawnie, np.: `.3[w,b]` jest równoważne wyrażeniu `w+.3(b-w)`, gdzie dopuszczalne

[5] Oznaczenie *bp* oznacza duży punkt (ang. *big point*) równy $\frac{1}{72}$ cala. Jest to jedna ze standardowych jednostek TEX-a i METAFONT-a, domyślnie używana przez MetaPost. Pełna lista wszystkich jednostek znajduje się na str. 92. *The* METAFONT*book*.).

* Dokładne informacje o tym, jak znaki `%` i `"` są traktowane przez METAFONT w procesie zamiany na żetony, znajdują na str. 50. *The* METAFONT*book*. (przyp. tłum.).

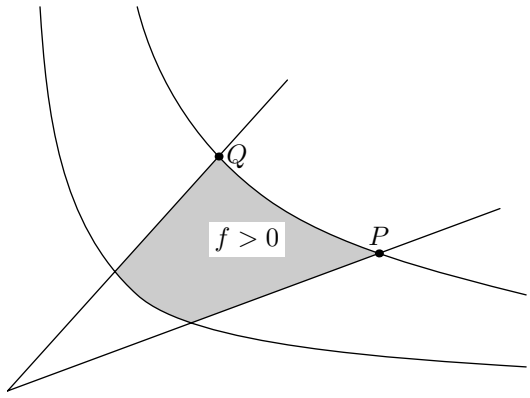
jest by `w` i `b` były m.in. zmiennymi typu `color`. Kolory są określone za pomocą predefiniowanych stałych `black`, `white`, `red`, `green`, `blue`, lub mogą być zdefiniowane przez użytkownika jako składowe RGB (R-ed, G-reen, B-lue). Czarny kolor jest zapisany jako `(0,0,0)`, a biały `(1,1,1)`. Nie ma żadnych ograniczeń w definiowaniu kolorów „czarniejszych niż czarny" i „bielszych niż biel". Przy definiowaniu pozostałych kolorów, należy użyć liczb z przedziału [0,1].

MetaPost rozwiązuje równania liniowe na zmiennych typu `color` w taki sam sposób, jak na zmiennych typu `pair` (patrz Rozdział 9. *The* METAFONT*book*).

Przyjrzyjmy się teraz przykładowi, który ilustruje niektóre z powyższych własności MetaPost-a.

```
beginfig(2);
h=2in; w=2.7in;
path p[], q[], pp;
for i=1.5,2,4: ii := i**2;
  p[i] = (w/ii,h){1/ii,-1}...(w/i,h/i)...(w,h/ii){1,-1/ii};
endfor
for i=.5,1.5: q[i] = origin..(w,i*h) cutafter p1.5; endfor
pp = buildcycle(q0.5, p2, q1.5, p4);
fill pp withcolor .8white;
z0=center pp;
picture lab; lab=thelabel(btex $f>0$ etex, z0);
unfill bbox lab; draw lab;
draw q0.5; draw p2; draw q1.5; draw p4;
makelabel.top(btex $P$ etex, p2 intersectionpoint q0.5);
makelabel.rt(btex $Q$ etex, p2 intersectionpoint q1.5);
endfig;
```

generuje następujący rysunek:



Trzecia linia deklaruje tablice ścieżek `p` i `q` zgodnie z opisem zawartym w Rozdziale 7. Zauważmy, że `q1.5` oznacza to samo co `q[i]`, dla `i`=1.5. Pętla `for` (dla każdego `i`) oblicza kolejne aproksymacje łuku hiperboli

$$xy = \frac{\omega h}{i^2}$$

a `q[i]` fragmenty prostej ih/ω (patrz Rozdział 19. *The* METAFONT*book*).

Operator `cutafter` jest użyty do odcięcia części krzywej `q[i]` za punktem przecięcia ze ścieżką `p1.5` (w pliku źródłowym nie ma napisu `draw p1.5`, który narysowałby ścieżkę `p1.5`, stąd też hiperbola ta nie jest widoczna na rysunku). Istnieje też operator `cutbefore`

$$a \texttt{ cutbefore } b$$

który pozostawi ze ścieżki a tylko ten fragment, który występuje za punktem przecięcia ze ścieżką b. W przypadku wielokrotnego przecinania się ścieżek, polecenia `cutbefore` oraz `cutafter` próbują odciąć najmniej jak jest to możliwe.

Zacieniowany obszar na przedstawionym rysunku otrzymany został w wyniku napisania polecenia

```
fill pp withcolor .8white
```

Granice zacieniowania określa ścieżka `pp`, którą tworzy makro `buildcycle` łączące kawałki czterech ścieżek podanych jako argumenty. Innymi słowy `pp` jest skonstruowana jako krzywa zamknięta, która najpierw podąża wzdłuż ścieżki `q0.5` do punktu przecięcia ze ścieżką `p2`, dalej wzdłuż `p2` do punktu przecięcia z `q1.5`, itd. W powyższym przykładzie narzucone wymagania powodują poruszanie się w kierunku przeciwnym do kierunku krzywych `p2` i `q1.5`. Zazwyczaj makro `buildcycle` próbuje unikać poruszania „pod prąd", jeśli tylko ma do wyboru więcej niż jeden punkt przecięcia (w tym przykładzie pary kolejnych ścieżek przecinają się dokładnie raz). Działanie tego makra jest przewidywalne tylko wówczas, gdy podane ścieżki przecinają się dokładnie w jednym punkcie. W przypadku, gdy występuje kilka punktów przecięcia, rezultat może być zaskakujący.

MetaPost-owe makra `fill` oraz `unfill` dają podobny efekt do odpowiednich makr METAFONT-owych, opisanych w Rozdziale 13. *The* METAFONT*book*, jednak ich realizacja przebiega zupełnie inaczej (wynika to z obwiedniowego charakteru MetaPost). Ponadto w MetaPost-cie nie występują polecenia `cull` oraz `withweight`. Makro `unfill` użyte jest do wymazania białego prostokąta, zawierającego napis „$f > 0$". Na powyższym rysunku działa prawidłowo dzięki domyślnemu użyciu `withcolor background` (w kolorze tła), gdzie kolorem tła jest kolor biały. Poniższy diagram składniowy przedstawia budowę podstawowych poleceń służących w MetaPost-cie do rysowania:

⟨rysuj⟩ → ⟨dorysuj⟩ | ⟨odetnij⟩
⟨dorysuj⟩ →
  `addto` ⟨zmienna typu `picture`⟩ `also` ⟨wyrażenie typu `picture`⟩ ⟨opis⟩
  | `addto` ⟨zmienna typu `picture`⟩ `contour` ⟨wyrażenie typu `path`⟩ ⟨opis⟩
  | `addto` ⟨zmienna typu `picture`⟩ `doublepath` ⟨wyrażenie typu `path`⟩ ⟨opis⟩
⟨opis⟩ → ⟨nic⟩ | ⟨warunek⟩ ⟨opis⟩
⟨warunek⟩ → `withcolor` ⟨wyrażenie typu `color`⟩
  | `withpen` ⟨wyrażenie typu `pen`⟩ | `dashed` ⟨wyrażenie typu `picture`⟩
⟨odetnij⟩ → `clip` ⟨wyrażenie typu `picture`⟩ `to` ⟨wyrażenie typu `path`⟩

Jeśli P oznacza bieżący element rysunku (`currentpicture`), q oznacza bieżące piórko (`currentpen`), a b kolor tła (`background`), wówczas standardowe makra służące do rysowania w przybliżeniu oznaczają (por. *The* METAFONT*book*, str. 118):

`draw` p	oznacza	`addto` P `doublepath` p `withpen` q
`fill` c	oznacza	`addto` P `contour` c
`filldraw` c	oznacza	`addto` P `contour` c `withpen` q
`undraw` p	oznacza	`addto` P `doublepath` p `withpen` q `withcolor` b
`unfill` c	oznacza	`addto` P `contour` c `withcolor` b
`unfilldraw` c	oznacza	`addto` P `contour` c `withpen` q `withcolor` b

Wyrażenie oznaczone w powyższej tabeli przez c musi być zamkniętą ścieżką, natomiast p nie. Możliwe jest też użycie poleceń `draw` oraz `undraw` kiedy argumentem jest rysunek r:

`draw` r	oznacza	`addto` P `also` r
`undraw` r	oznacza	`addto` P `also` r `withcolor` b

W ostatnim przykładzie argumentem polecenia `unfill` jest `bbox lab`. Jest to wywołanie standardowego makra tworzącego ścieżkę w kształcie prostokątnej ramki (ang. *bounding box*). Makro `center` użyte dwie linie wcześniej określa punkt `z0` jako środek ścieżki `pp` (to makro można użyć zarówno dla wyrażeń typu `path`, jak i `picture`). Wyrażenie

```
thelabel(btex $f>0$ etex, z0)
```

generuje obrazek z napisem „$f > 0$”, wyśrodkowany w punkcie `z0`.

Niżej jest przedstawiona składnia poleceń wprowadzających napisy:

⟨napisz⟩ → ⟨jak napisać⟩⟨jak przyczepić⟩(⟨napis⟩, ⟨punkt przyczepienia⟩)
| `labels`⟨jak przyczepić⟩(⟨ciąg argumentów⟩)
⟨jak napisać⟩ → `label` | `thelabel` | `makelabel`
⟨jak przyczepić⟩ → ⟨nic⟩ | `.lft` | `.rt` | `.top` | `.bot`
| `.ulft` | `.urt` | `.llft` | `.lrt`
⟨napis⟩ → ⟨wyrażenie typu `picture`⟩ | ⟨wyrażenie typu `string`⟩
⟨punkt przyczepienia⟩ → ⟨wyrażenie typu `pair`⟩
⟨ciąg argumentów⟩ → ⟨argument⟩ | ⟨argument⟩, ⟨ciąg argumentów⟩

Polecenie `label` dołącza napisy do bieżącego rysunku w miejscu określonym przez ⟨punkt przyczepienia⟩ i ⟨jak przyczepić⟩. Użycie ⟨nic⟩ powoduje wyśrodkowanie wprowadzanego obiektu, przesuwając go nieznacznie tak, by nie przykrył punktu określonego przez ⟨punkt przyczepienia⟩. Użycie polecenia `thelabel` spowoduje wygenerowanie wyrażenia typu `picture` i ustalenie go jako rysunek bieżący (`currentpicture`). Użycie `makelabel` zamiast `label` spowoduje dodanie punktu w miejscu określonym przez ⟨punkt przyczepienia⟩. Polecenie `labels` jest jednoznaczne z poleceniem

`makelabel`⟨jak przyczepić⟩(`str`⟨argument⟩, `z`⟨argument⟩)

dla każdego ⟨argumentu⟩ z ⟨ciągu argumentów⟩. Użycie operatora `str` powoduje zmianę typu argumentu na łańcuchowy (`string`). Stąd `labels.top(1,2a)` umieści napisy „`1`” oraz „`2a`” odpowiednio nad punktami `z1` oraz `z2a`.

Przykłady przedstawione dotychczas wprowadzają ⟨napis⟩ za pomocą konstrukcji

`btex` ⟨polecenia TEX-a⟩ `etex`

który w ten sposób jest przekształcany w wyrażenie typu `picture`. Jeśli wprowadzany napis jest wystarczająco prosty, może być umieszczony na rysunku jako wyrażenie typu `string`, bieżącym fontem (`defaultfont`) i w bieżącej skali (`defaultscale`). Domyślne wartości:

`defaultfont="cmr10"` i `defaultscale=1`.

Wartości tych parametrów można dowolnie zmieniać. Na przykład wpisanie `cmtex10` zamiast `cmr10` pozwala na użycie we wprowadzanym napisie spacji oraz znaków specjalnych. Jeśli powstaną wątpliwości co do skali, wygodnie jest posłużyć się poleceniem `fontsize`

`defaultfont="Times"; defaultscale:=10/fontsize "Times".`

Zwróćmy uwagę na ⟨warunek⟩ określony jako `dashed` ⟨wyrażenie typu `picture`⟩. W ten sposób można rysować linią przerywaną. Pod poleceniem `evenly` kryje się standardowa definicja linii przerywanej, jako kreski o długości $3bp$ z odstępem o długości $3bp$. Oczywiście możliwe jest przeskalowanie `evenly` w celu zagęszczenia, bądź też rozrzedzenia wzoru. Tak więc polecenie

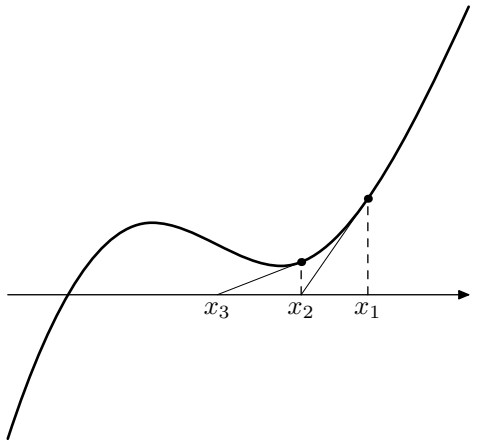
`draw z1..z2 dashed evenly scaled 2`

powoduje narysowanie linii przerywanej kreskami o długości $6bp$ z przerwami co $6bp$.

Następujący przykład ilustruje sposób użycia linii przerywanych:

```
beginfig(3);
3.2scf = 2.4in;
path fun;
# = .1;
fun = ((0,-1#)..(1,.5#){right}..(1.9,.2#){right}
       ..{curl .1}(3.2,2#))  scaled scf yscaled(1/#);
vardef vertline primary x = (x,-infinity)..(x,infinity) enddef;
primarydef f atx x = (f intersectionpoint vertline x) enddef;
primarydef f whenx x = xpart(f intersectiontimes vertline x)
  enddef;
z1a = (2.5scf,0);
z1 = fun atx x1a;
y2a=0; z1-z2a=whatever*direction fun whenx x1 of fun;
z2 = fun atx x2a;
y3a=0; z2-z3a=whatever*direction fun whenx x2 of fun;
draw fun withpen pencircle scaled 1pt;
drawarrow (0,0)..(3.2scf,0);
label.bot(btex $x_1$ etex, z1a);
draw z1a..z1 dashed evenly;
makelabel(nullpicture, z1);
draw z1..z2a withpen pencircle scaled .3;
label.bot(btex $x_2$ etex, z2a);
draw z2a..z2 dashed evenly;
makelabel(nullpicture, z2);
draw z2..z3a withpen pencircle scaled .3;
label.bot(btex $x_3$ etex, z3a);
endfig;
```

Na podstawie powyższego programu MetaPost wygeneruje następujący rysunek:



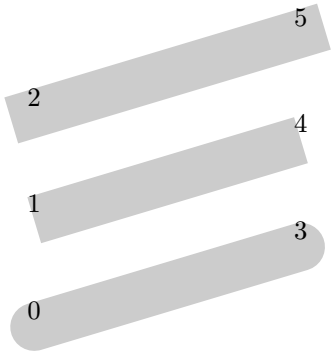
W powyższym przykładzie wykorzystywane są bardziej zaawansowane właściwości ścieżek, opisane w Rozdziale 14. *The* METAFONT*book*. W zasadzie cały ten rozdział – mimo, iż jest poświęcony METAFONT-owi – można odnieść do MetaPost-a, z wyjątkiem zastosowania „dziwnych ścieżek" (ang. *strange path*), które w MetaPost-cie nie występują. W celu zrozumienia w jaki sposób powstał powyższy rysunek, konieczna jest znajomość związków pomiędzy operatorami oraz znaczenia poleceń `direction`, `intersectiontimes` i `intersectionpoint`. W powyższym przykładzie przyjęto, że ścieżka `fun` czyni y funkcją zmiennej x. Funkcja jest konstruowana ze współrzędną y przemnożoną przez `.1` (czyli podzieloną przez 10). Po obliczeniu przez MetaPost współrzędnych szukanych punktów polecenie „`yscaled(1/#)`" przywraca zmiennej y oryginalny wymiar.

Polecenie `yscaled` jest przykładem polecenia należącego do bardzo ważnej klasy operatorów, odpowiadających za przekształcenia afiniczne punktów (`pair`), ścieżek (`path`), piórek (`pen`), obrazków (`picture`) oraz innych obiektów. Jedyną różnicą między właściwościami METAFONT-a opisanymi w Rozdziale 15., a MetaPost-em jest to, że MetaPost nie posiada polecenia *currenttransform* oraz ograniczeń co do rodzaju przekształceń jakie można wykonać na danym obiekcie.

3. Zagadnienia bardziej zaawansowane

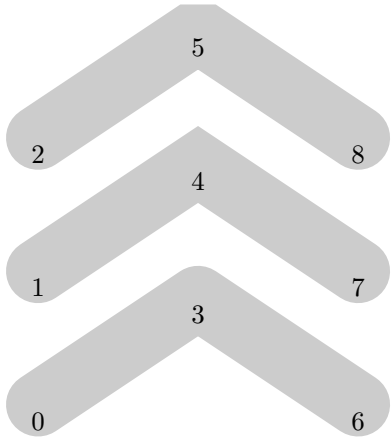
MetaPost posiada możliwość zdefiniowania piórka w taki sam sposób, jak METAFONT. Dla zwykłego użytkownika najważniejsza jest możliwość zmiany szerokości linii, jak zostało to przedstawione w ostatnim przykładzie. Jako uwagę do Rozdziału 16. *The* METAFONT*book* można dodać, że MetaPost nie posiada czegoś takiego jak „future pen" oraz że eliptyczne piórko nie da się przekształcić w wielokąt. Ponadto w MetaPost-cie nie są potrzebne operatory `cutoff` oraz `cutdraw`, ponieważ ten sam efekt może być osiągnięty przez ustawienie `linecup:=butt`.

```
beginfig(4);
for i=0 upto 2:
  z[i]=(0,40i); z[i+3]-z[i]=(100,30);
endfor
pickup pencircle scaled 18;
def gray = withcolor .8white enddef;
draw z0..z3 gray;
linecap:=butt; draw z1..z4 gray;
linecap:=squared; draw z2..z5 gray;
labels.top(0,1,2,3,4,5);
endfig; linecap:=rounded;
```



W MetaPost-cie dostępny jest też parametr `linejoin` zilustrowany poniżej. Domyślną wartością `linecap` oraz `linejoin` jest `rounded`.

```
beginfig(5);
for i=0 upto 2:
  z[i]=(0,50i); z[i+3]-z[i]=(60,40);
  z[i+6]-z[i]=(120,0);
endfor
pickup pencircle scaled 24;
def gray = withcolor .8white enddef;
draw z0--z3--z6 gray;
linejoin:=mitered; draw z1..z4--z7 gray;
linejoin:=beveled; draw z2..z5--z8 gray;
labels.bot(0,1,2,3,4,5,6,7,8);
endfig; linejoin:=rounded;
```



Innym sposobem na narysowanie pożądanej linii jest użycie konstrukcji

`drawoptions`(⟨*opis*⟩)

Na przykład polecenie

```
drawoptions(withcolor blue)
```

sprawi, że dla reszty rysunku bieżącym kolorem będzie niebieski. To ustawienie może być lokalnie zmienione przez `withcolor`

```
draw p withcolor red.
```

Polecenie to może występować także w połączeniu z poleceniem rysującym

```
drawoptions (dashed dd)
```

i wówczas działać będzie jak polecenie `draw`, ale nie będzie wypełniać konturów.

Rozdziały 17–20 *The* METAFONT*book* opisują konstrukcje języka jakie można stosować do rozwiązywania poszczególnych problemów. Niektóre opisane właściwości METAFONT-a, w odniesieniu do MetaPost-a wymagają dokładniejszego omówienia. Nie wszystkie wymienione w tych rozdziałach polecenia występujące w plainowych makrach METAFONT-a, występują w makrach MetaPost-a

Polecenia `beginfig` oraz `endfig` odgrywają taką rolę w MetaPost-cie, jak polecenia `beginchar` oraz `endchar` w METAFONT-cie. Jeśli masz wątpliwości co do predefiniowanych makr, zajrzyj do definicji standardowych makr w pliku `plain.mp`. Ten plik może być także dobrym źródłem przykładów.

Rozdział 17. przedstawia jak wyrażenie (ang. *statement*) `interim` powoduje chwilowe zmiany wewnętrznych wielkości (ang. *internal quantities*). Podobnie działa to w MetaPost-cie. Wyjątkiem jest przykład wykorzystujący wartość *autorounding*, ponieważ MetaPost nie posiada tej szczególnej wielkości. Poniżej znajduje się pełny wykaz wielkości występujących w METAFONT-cie, a nie występujących w MetaPost-cie:

```
autorounding, fillin, granularity, hppp, proofing,
smoothing, tracingedges, tracingpens, turningcheck,
vppp, xoffset, yoffset
```

Następujące dodatkowe wielkości są zdefiniowane w makrach plainowych języka METAFONT, a nie występują w makrach MetaPost-a:

```
pixels_per_inch, blacker, o_correction, displaying,
screen_rows, screen_cols, currentwindow
```

Występują też takie, które są dostępne tylko w MetaPost-cie. Polecenia `linecap` oraz `linejoin` zostały już opisane. Występuje także polecenie `miterlimit`, które zachowuje się tak samo, jak podobnie nazwane polecenie w języku PostScript. Parametr `tracing_lost_chars` wyłącza komunikaty o błędach, przy próbach użycia nieistniejących znaków (ang. *missing characters*). Jest to prawdopodobnie jedyny przypadek, kiedy niemożliwe jest użycie nieistniejących znaków w łańcuchu tekstowym, wprowadzanym za pomocą polecenia `label` w bloku `btex ... etex`.

Parametr `prologues` został już opisany, kiedy zalecaliśmy nadanie mu wartości 1, w przypadku, gdy mamy do czynienia z dokumentem *troff*-a. Dowolna dodatnia wartość będzie powodować, że plik wyjściowy będzie zgodny z PostScript-em, tzn. zostaną użyte standardowe fonty Postscriptowe Adobe Type 1. Dzięki temu plik będzie bardziej „przenośny", ale dla większości aplikacji (do których wprowadzimy plik wyjściowy MetaPost-a) niemożliwe będzie użycie TEX-owych fontów, takich jak np. `cmr10`.* Drukarki i inne urządzenia, które potrafią interpretować język PostScript, nie posiadają wbudowanych fontów z rodziny CM.

MetaPost-owy plain udostępnia nam także takie wewnętrzne parametry, jak `bboxmargin`, `labeloffset` oraz `ahangle`, tego samego typu, co już opisany parametr `defaultscale`, kontrolujący rozmiar używanego fontu. Parametr `bboxmargin` przekazuje informację o wielkości dodatkowej przestrzeni, pozostawionej przez operator `bbox`; `labeloffset` określa odległość o jaką jest przesunięty wprowadzany tekst względem punktu położenia etykiety; `ahangle` jest kątem między ramionami grotu strzałki (`drawarrow`) – domyślnie wynosi 45°. Występuje też ścieżka `ahcirc`, która określa rozmiar grotu strzałki. Konstrukcja

```
ahcirc:=fullcircle scaled d
```

* Sytuacja ta uległa zmianie po pojawieniu się fontów z rodziny CM w postaci fontów postscriptowych Typu 1. (przyp. tłum.)

zmienia długość grotu strzałki na `d/2`.

Partią materiału opisaną w *The* METAFONT*book*, która nie zostanie wyjaśniona w dalszej części, jest zawartość Rozdziałów 21–22 oraz Dodatku D. Rozdziały 23 i 24 w ogóle nie mają odzwierciedlenia w MetaPost-cie. Budowa składni podana w Rozdziałach 25 i 26 nie jest identyczna z budową składni MetaPost-a, ale większość różnic została już opisana. Dodatkowo występują operatory `redpart`, `bluepart` i `greenpart`, a nie występuje `totalweight`. Nowością służącą do wprowadzania tekstów w rysunkach jest polecenie

⟨drugorzędny rysunek⟩ → ⟨drugorzędny rysunek⟩ `infont` ⟨łańcuch tekstu⟩

Informacje o rozmiarze rysunku (ang. *bounding box*) mogą być przekazane za pomocą parametrów

⟨współrzędna⟩ → ⟨narożnik⟩⟨bieżący rysunek⟩
⟨narożnik⟩ → `llcorner | urcorner | lrcorner | urcorner`

Głównym zastosowaniem powyższej konstrukcji w MetaPost-cie jest umieszczanie tekstu w rysunkach; może być też ona stosowana w rysunkach zawierających dowolne mieszaniny tekstu i grafiki.

Polecenie

`special` ⟨wyrażenie typu `string`⟩

dodaje linię tekstu na początku bieżącego pliku wyjściowego. W ten sposób można na przykład dodać PostScript-owe definicje, które sprawią, że plik wyjściowy MetaPost-a będzie zawierał odwołanie do fontu `Times-Roman`.

```
special "/Times-Roman /Times-Roman def";
special "/fshow {exch findfont exch scalefont setfont show}";
special " bind def";
```

Podobna definicja jest tworzona automatycznie, kiedy używamy polecenia `prologues:=1`. Gdy `prologues` jest równe zero, program, który łączy plik wyjściowy w TEX-u z rysunkami w PostScript-cie będzie dodawał potrzebne definicje.[6]

Inną nowością MetaPost-a, która wymaga dodatkowego wyjaśnienia jest pomysł zdefiniowania przerywanej linii (ang. *dash pattern*). Jeśli tylko można, najłatwiej jest posłużyć się standardowym wzorcem, który jest zdefiniowany jako `evenly`. Czasami jednak okazuje się konieczne podanie odpowiedniej reguły, która spowoduje wygenerowanie linii o takim, a nie innym wyglądzie.

Linia przerywana jest obrazkiem zawierającym jedną lub więcej poziomych kawałków linii. Jest bez znaczenia, jakie piórko zostanie użyte do ich narysowania. MetaPost zachowuje się tak, jakby podany wzór odzwierciedlał nieskończenie długą, poziomą, przerywaną linię, użytą jako wzorzec linii przerywanej. Na przykład następujące polecenia tworzą wzór linii przerywanej `dd`:

```
draw (1,0)..(3,0); draw (5,0)..(6,0);
```

[6] Pełny opis jak wpływać na zawartość pliku wyjściowego dokumentu napisanego w TEX-u, znajduje się w dalszej części tej dokumentacji. Plik wyjściowy wygenerowany przez MetaPost, gdy `prologues` jest równe jeden, może być wysłany do drukarki PostScript-owej wtedy, gdy odwołuje się on do jakiegoś wbudowanego fontu drukarki np. `Helvetica`.

```
picture dd; dd:=currentpicture; clearit;
```

Ustawienie w rządku nieskończonej liczby kopii `dd`, spowoduje powstanie zbioru fragmentów linii

$$\{(5i,0)...(5i+3,0) \mid i \text{ należy do liczb całkowitych } (i \in Z)\}.$$

Edycja wzoru rozpoczyna się od osi y i porusza się w prawo; rysowane są linie o długości $3bp$, oddzielane przerwami o długości $2bp$.

W tym przykładzie kolejne kopie `dd` są przesunięte o $5bp$, ponieważ zasięg współrzędnej x w `dd` rysuje fragment przerywanej linii o długości $6-1$, czyli $5bp$. Przesunięcie to może być powiększone przez zwiększenie współrzędnej y tak, by jej wartość bezwzględna była większa niż $5bp$. Regułą jest, że poziomy odstęp między kopiami fragmentów przerywanych linii jest równy maksimum z $|y|$ oraz zasięgu współrzędnej x.

Pudełka

W pakiecie MetaPost-a znajdują się też pomocnicze makra, które nie są zawarte w bazie plain. Znajdują się one na przykład w pliku `boxes.mp`. Zawartość tego podrozdziału poświęcona jest właśnie makrom z tego pakietu. Makra te mogą być interesujące dla użytkowników, którzy szukają dodatkowych przykładów możliwości, jakie oferuje program MetaPost.

Najważniejszym makrem jest

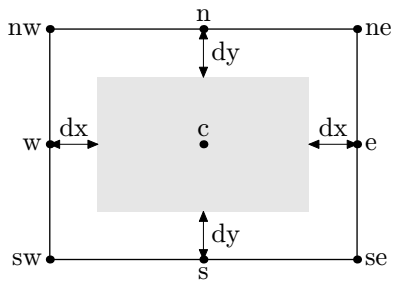
`boxit`⟨argument⟩(⟨wyrażenie typu `picture`⟩)

znajdujące wartości zmiennych typu `pair`: ⟨argument⟩`.c`, ⟨argument⟩`.n`, ⟨argument⟩`.e` itd., które mogą być użyte do określenia położenia rysunku, zanim zostanie on narysowany za pomocą specjalnego polecenia rysującego:

`drawboxed`(⟨argument⟩)

Polecenie `boxit.bb(pic)` ustali zmienną `bb.c` jako miejsce, gdzie zostanie położony środek obrazka `pic`. Zdefiniowane zostaną wartości `bb.sw`, `bb.se`, `bb.ne`, i `bb.nw`, które określą położenie wierzchołków prostokątnej ścieżki, która ostatecznie będzie go otaczać. Zmienne `bb.dx` oraz `bb.dy` przechowują informacje o odstępach między ostateczną wersją obrazka (`pic`), a bokami otaczającego go prostokąta.

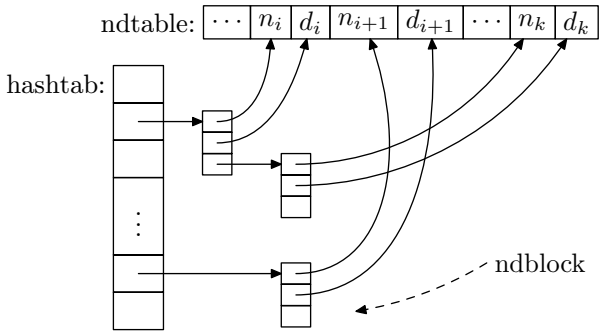
Makro `boxit` tworzy liniowe równanie, którego rozwiązaniem są `bb.sw`, `bb.se`, ... będące wierzchołkami prostokąta we współrzędnych prostokątnych. Wewnątrz tego prostokąta znajdzie się wyśrodkowany obrazek. Wartości `bb.dx`, `bb.dy`, `bb.c` są nieokreślone. Odpowiednie równanie, decydujące o położeniu pudełka, może być podane przez użytkownika. Jeśli równanie to nie jest podane, makra takie jak na przykład `drawbox` przyjmują wartości domyślne.



Następujący przykład demonstruje, jak to wygląda w praktyce.

```
beginfig(7); boxjoin(a.se=b.sw; a.ne=b.nw );
boxit.a(btex $\cdots$ etex);    boxit.ni(btex $n_i$ etex);
boxit.di(btex $d_i$ etex);      boxit.nii(btex $n_{i+1}$ etex);
boxit.dii(btex $d_{i+1}$ etex); boxit.aa(pic_.a);
boxit.nk(btex $n_k$ etex);      boxit.dk(btex $d_k$ etex);
di.dy = 2;
drawboxed(a,ni,di,nii,dii,aa,nk,dk); label.lft("ndtable:", a.w);
boxjoin(a.sw=b.nw; a.se=b.ne);
interim defaultdy:=7;
boxit.ba(); boxit.bb(); boxit.bc();
boxit.bd(btex $\vdots$ etex); boxit.be(); boxit.bf();
bd.dx = 8; ba.ne = a.sw - (15,10);
drawboxed(ba,bb,bc,bd,be,bf); label.lft("hashtab:", ba.w);
def ndblock suffix $ =
  boxjoin(a.sw=b.nw; a.se=b.ne);
  forsuffixes $$=$a,$b,$c: boxit$$(); ($$dx,$$dy)=(5.5,4);
  endfor; enddef;
ndblock nda; ndblock ndb; ndblock ndc;
nda.a.c - bb.c = ndb.a.c - nda.c.c = (whatever,0);
xpart ndb.c.se = xpart ndc.a.ne = xpart di.c;
ndc.a.c - be.c = (whatever,0);
drawboxes(nda.a,nda.b,nda.c,ndb.a,ndb.b,ndb.c,ndc.a,ndc.b,ndc.c);
drawarrow bb.c .. nda.a.w;
drawarrow be.c .. ndc.a.w;
drawarrow nda.c.c .. ndb.a.w;
drawarrow nda.a.c{right}..{curl0}ni.c cutafter bpath ni;
drawarrow nda.b.c{right}..{curl0}di.c cutafter bpath di;
drawarrow ndc.a.c{right}..{curl0}nii.c cutafter bpath nii;
drawarrow ndc.b.c{right}..{curl0}dii.c cutafter bpath dii;
drawarrow ndb.a.c{right}..nk.c cutafter bpath nk;
drawarrow ndb.b.c{right}..dk.c cutafter bpath dk;
x.ptr = xpart aa.c; y.ptr = ypart ndc.a.ne;
drawarrow subpath (0,.7) of (z.ptr..{left}ndc.c.c) dashed evenly;
label.rt(btex ndblock etex, z.ptr); endfig;
```

Pouczające jest porównanie MetaPost-owego rysunku umieszczonego poniżej z odpowiednim rysunkiem wykonanym za pomocą języka *pic*, zawartymi w podręczniku [2].



Druga linia kodu generującego powyższy rysunek zawiera polecenie

```
boxjoin(a.se=b.sw; a.ne=b.nw)
```

Dodatkowe równanie użyte jako argument powoduje, w tym przypadku, umieszczanie pudełek w linii poziomej dopóty, dopóki po pudełku `a` występuje jakieś pudełko `b`. Pierwsze wywołanie równania występuje w następnej linii, gdzie po pudełku `a` umieszczone jest pudełko `ni`. Czyli

```
a.se=ni.sw; a.ne=ni.nw
```

Następną parą są pudełka `ni` oraz `di`. Tym razem niejawnie wygenerowanymi równaniami są:

```
ni.se=di.sw; ni.ne=di.nw
```

Ten proces jest kontynuowany, aż zostanie podane nowe polecenie `boxjoin`. W tym przykładzie nową deklaracją jest

```
boxjoin(a.sw=b.nw; a.se=b.ne),
```

która powoduje, że pudełka są umieszczane pionowo jedno pod drugim.

Po wywołaniu `boxit` dla pierwszych ośmiu pudełek, od `a` do `dk`, występuje równanie `di.dy = 2`. Poprzedza ono wywołanie makra `drawboxed`, które rysuje te pudełka, umieszczając wewnątrz każdego odpowiedni tekst. Równanie to wymusza, by odstępy nad i pod zawartością pudełka `di` (z tekstem d_i) wynosiły $2bp$, nie daje jednak pełnego określenie rozmiaru i położenia. Makro `drawboxed` rozpoczyna więc działanie od przypisania domyślnej wartości $3bp$ zmiennym `a.dx ... dk.dx`.

Argument makra `boxit` może być opuszczony, jak w powyższym przykładzie np. `boxit.ba()` czy `boxit.bb()`. Takie wywołanie jest jednoznaczne z wywołaniem makra `boxit` bez rysunku. Zamiast rysunku, argumentem może być wyrażenie typu `string`. W tym przypadku wprowadzany napis tworzony jest bieżącym fontem.

Dodatkowo, dla punktów określających wierzchołki `a.sw`, `a.se`, ..., polecenie `boxit.a` definiuje punkty `a.w`, `a.s`, `a.e` oraz `a.n`, które są środkami odcinków, z których zbudowany jest prostokąt. Jeśli potrzebujemy inny prostokąt, niż rysowany przez makro `drawboxed`, to do określenia własnego prostokąta można użyć makro `bpath.a`. Ogólnie:

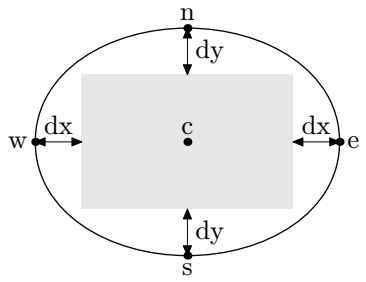
```
bpath(nazwa pudełka)
```

W powyższym przykładzie makro `bpath` jest użyte jako argument polecenia `drawarrow`. Dla przykładu

```
nda.a.cright..curl0ni.c
```

jest ścieżką przebiegającą od środka pudełka `nda` do środka pudełka `ni`. Kolejne polecenie „`cutafter bpath.ni`" powoduje, że grot strzałki skierowanej do środka pudełka `ni` zatrzymuje się w miejscu przecięcia z bokiem otaczającego prostokąta.

Następny przykład również używa techniki odcinania strzałki po przecięciu z otaczającą ścieżką, ale w tym przypadku otaczającą ścieżką nie jest prostokąt, lecz elipsa (może być też okrąg). Efekt taki uzyskujemy zastępując makro `boxit` makrem `circleit`. Pisząc `circleit.a(pic)` definiujemy punkty `a.c`, `a.s`, `a.e`, `a.n`, `a.w` oraz odstępy `a.dx` i `a.dy`. Zmienne te opisują położenie rysunku wewnątrz elipsy, a to w jaki sposób są użyte, przedstawia rysunek poniżej.

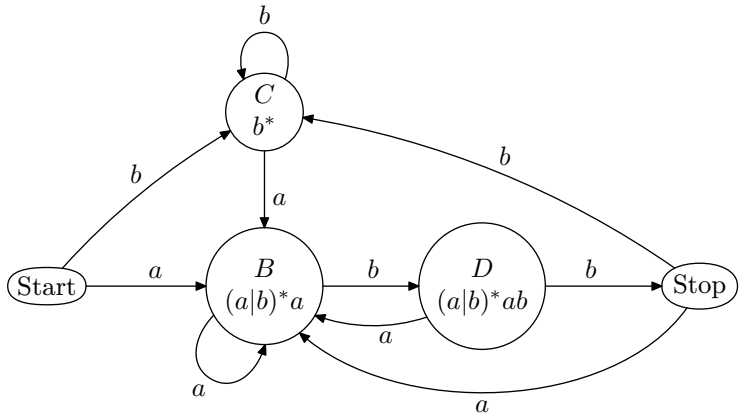


Poniższy kod rysunku zawiera przykład użycia makra `circleit`:

```
beginfig(9);
vardef cuta(suffix a,b) expr p =
  drawarrow p cutbefore bpath.a cutafter bpath.b;
  point .5*length p of p
enddef;
vardef self@# expr p =
  cuta(@#,@#) @#.c{curl0}..@#.c+p..{curl0}@#.c  enddef;
verbatimtex
  \def\stk#1#2{$\displaystyle{\matrix{#1\cr#2\cr}}$} etex
circleit.aa("Start"); aa.dx=aa.dy;
circleit.bb(btex \stk B{(a|b)^*a} etex);
circleit.cc(btex \stk C{b^*} etex);
circleit.dd(btex \stk D{(a|b)^*ab} etex);
circleit.ee("Stop"); ee.dx=ee.dy;
numeric hsep;
bb.c-aa.c = dd.c-bb.c = ee.c-dd.c = (hsep,0);
cc.c-bb.c = (0,.8hsep);
xpart(ee.e - aa.w) = 3.8in;
drawboxed(aa,bb,cc,dd,ee);
label.ulft(btex$b$etex, cuta(aa,cc) aa.c{dir50}..cc.c);
label.top(btex$b$etex, self.cc(0,30pt));
label.rt(btex$a$etex, cuta(cc,bb) cc.c..bb.c);
label.top(btex$a$etex, cuta(aa,bb) aa.c..bb.c);
label.llft(btex$a$etex, self.bb(-20pt,-35pt));
```

```
label.top(btex$b$etex, cuta(bb,dd) bb.c..dd.c);
label.top(btex$b$etex, cuta(dd,ee) dd.c..ee.c);
label.lrt(btex$a$etex, cuta(dd,bb) dd.c..{dir140}bb.c);
label.bot(btex$a$etex,
  cuta(ee,bb) ee.c..tension1.3 ..{dir115}bb.c);
label.urt(btex$b$etex,
  cuta(ee,cc) ee.c{(cc.c-ee.c)rotated-15}..cc.c);
endfig;
```

„Pudełka” uzyskiwane za pomocą polecenia `circleit`, zwykle są okręgami, chyba, że coś wymusi inny wygląd.



Na powyższym rysunku, równania `aa.dx = aa.dy` oraz `ee.dx = ee.dy` użyte po

`circleit.aa("Start")` oraz `circleit.ee("Stop")`

powodują, że węzły Start i Stop nie są okręgami.

Ogólną zasadą jest, że jeśli c`.dx`, c`.dy` oraz c`.dx`–c`.dy` nie są określone, wówczas `bpath.`c jest okręgiem. W przeciwnym wypadku makro tworzy owal wystarczająco duży, by pomieścić wewnątrz dany obrazek (wielkość marginesu jest przekazana za pomocą wewnętrznego parametru `circmargin`).

Występuje też makro `pic`, które tworzy rysunek `pic.`c wewnątrz ścieżki `bpath.`c. Poza makrem `drawboxed`, które rysuje rysunek wraz z otaczającym prostokątem lub owalem, dostępne są też makra `drawunboxed` oraz `drawboxes`, które rysują odpowiednio sam obrazek lub samą otaczającą ścieżkę.

4. Zakończenie

Interpreter MetaPost-a został napisany w języku `WEB` (stworzonym przez Donalda E. Knuth'a), który może być traktowany jak poszerzony o dodatkowe makra język PASCAL. Wybór ten pozwala na dzielenie kodu źródłowego METAFONT-a. Około trzech czwartych źródłowego pliku `mp.web` zostało, za zgodą Knuth'a, utworzone ze źródeł METAFONT-a.

Zgodnie ze standardową metodą stosowaną dla programów napisanych w języku `WEB`, część programu przeznaczona do działania w systemie UNIX, jest podana w oddzielnym pliku `mp.ch`. Specjalny procesor `tangle` łączy go z plikiem

`mp.web`, tworząc program PASCAL-owy (później automatycznie przekształcany na C, przy użyciu specjalnego translatora, który jest zawarty w UNIX-owej wersji TEX-a). Jedynymi dodatkami wymaganymi przez interpreter MetaPost-a są: krótki, zewnętrzny program w C `mpext.c` oraz mały wejściowy plik `mp.h`, łączący wszystko razem.

Dodatkowo dla głównego interpretera stworzone są programy sterujące procesem interpretacji poleceń umieszczonych w bloku `btex ... etex`. Kiedy interpreter napotyka napis `btex` w dowolnym pliku wejściowym `foo.mp`, generuje pomocniczy plik `foo.mpx`. Plik `foo.mpx` zawiera tłumaczenie zawartości bloku `btex ... etex` (z pliku `foo.mp`) na niskiego poziomu polecenia MetaPost-owe. Jeśli plik `foo.mp` nie posiada danych lub nie istnieje, interpreter MetaPost-a przywołuje skrypt, który go generuje.

Tworzenie pomocniczego pliku `foo.mpx` na podstawie `foo.mp` jest procesem trzystopniowym: najpierw program C uruchamia `mptotex`, który tłumaczy zawartość pliku na polecenia TEX-owe; następnie TEX tworzy plik binarny, zawierający instrukcje niskiego poziomu; na koniec `WEB`-owy program `dvitomp` zapisuje równoważne MetaPost-owe polecenia w pliku `foo.mpx`. Kiedy używamy *troff*-a, programy `mptotex` oraz `dvitomp` są zastąpione odpowiednio programami napisanymi w języku C: `mptotr` oraz `dmp`.

5. Literatura

[1] John D. Hobby. A METAFONT-like system with PostScript output. *Tugboat, the TEX User's Group Newsletter*, 10(4):505–512, December 1989.

[2] Brian W. Kernighan. Pic—a graphics language for typesetting. In *Unix Research System Papers, Tenth Edition*, pages 53–77. AT&T Bell Laboratories, 1990.

[3] Donald E. Knuth. *Computers and Typesetting*, volume C. Addison Wesley, Reading, Massachusetts, 1986.

[4] Donald E. Knuth. *Computers and Typesetting*, volume D. Addison Wesley, Reading, Massachusetts, 1986.

6. Nota tłumacza

Tytuł oryginału brzmi „The MetaPost System". John Hobby był bardzo zdziwiony, gdy zwracając się do niego o zgodę na publikację mojego tłumaczenia, podałam datę powstania „The MetaPost System" na luty 1997. Okazało się, że LaTeX automatycznie wstawił bieżącą datę do kompilowanego dokumentu, co było przyczyną nieporozumienia. Tak naprawdę „The MetaPost System" powstał w sierpniu 1990 przed napisaniem podręcznika „A User's Manual for MetaPost".

Oto oryginalny fragment listu Jona Hobby, w którym zgadza się na publikację tego tłumaczenia:

[...] *It is all right with me if you post a translation of "The MetaPost System", but please make it clear that this document was written in 1990 and predates "A User's Manual for MetaPost".*

- John Hobby

Po raz pierwszy tłumaczenie ujrzało świat na V Konferencji BachoTeX w maju 1997 r. Zastrzegam sobie wszelkie prawa autorskie do tłumaczenia. Zgadzam się na dowolne rozpowszechnianie tego artykułu w postaci pliku postscriptowego i PDF, który umieszczony jest w archiwum GUST-u pod adresem:
`ftp.gust.org.pl/GUST/contrib/docs/mpintropl.ps`

Na zakończenie pragnęłabym gorąco podziękować za inspirację oraz pomoc merytoryczną B. Lichońskiemu oraz S. Wawrykiewiczowi za ostateczny szlif. Zdaję sobie sprawę z pewnych niedociągnięć, a być może nawet pomyłek. Jest to wersja 1.0, dlatego też byłabym wdzięczna za wszelkie uwagi.

Joanna Marszałkowska `joanna@ksinet.univ.gda.pl`