

Zpravodaj Československého sdružení uživatelů TeXu

Miroslava Krátká

METAPOST a mfpic - první část

Zpravodaj Československého sdružení uživatelů TeXu, Vol. 11 (2001), No. 1-3, 40–65

Persistent URL: <http://dml.cz/dmlcz/150209>

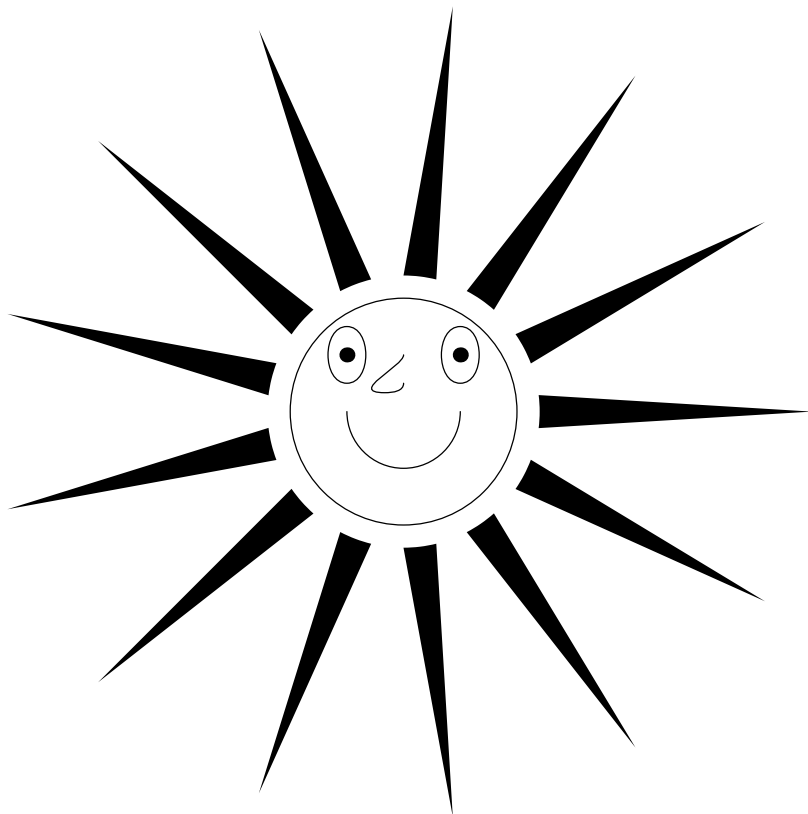
Terms of use:

© Československé sdružení uživatelů TeXu, 2001

Institute of Mathematics of the Czech Academy of Sciences provides access to digitized documents strictly for personal use. Each copy of any part of this document must contain these *Terms of use*.



This document has been digitized, optimized for electronic delivery and stamped with digital signature within the project *DML-CZ*:
The Czech Digital Mathematics Library <http://dml.cz>



Obsah

1	METAPOST	41
1.1	Datové typy	41
1.2	Vstupní soubor	43
1.3	Základní příkazy kreslení	45
1.4	Vkládání textu a obrázků	51
1.5	Výplně	55
1.6	Makra	56
1.7	Prologues	62
1.8	METAFONT a METAPOST	62
Summary: METAPOST and mfpic—the first part		65

1. METAPOST

METAPOST je programovací jazyk určený pro popis (a následné kreslení) obrázků. Jeho autorem je John D. Hobby. METAPOST vznikl z jazyku METAFONT, který již v roce 1977 začal vytvářet společně s typografickým systémem \TeX Donald E. Knuth. První zmínka o METAPOSTu se objevila v roce 1989.

Zdrojový text METAPOSTu je posloupnost příkazů oddělených středníky. Přesnější syntaxe souboru bude vysvětlena později. Překladač mívá obvykle podobný název, v operačním systému Linux je to `mpost`. Výstupem překladače je program v jazyce PostScript vykreslující obrázek.

1.1. Datové typy

Každý objekt v METAPOSTu má svůj datový typ. METAPOST podporuje následující datové typy:

1. `numeric` pro uložení čísla,
2. `pair` pro uložení souřadnic,
3. `path` pro uložení cesty, to jest křivky,
4. `transform` pro uložení afinní transformace,
5. `color` pro uložení barvy,
6. `string` pro uložení řetězce,
7. `boolean` pro uložení proměnné booleovského typu,
8. `picture` pro uložení obrázku,
9. `pen` pro uložení pera.

Čísla jsou reprezentována jako k -násobky zlomku $\frac{1}{65536}$, kde k je celé číslo. Jejich absolutní hodnota musí být menší než 4096, ale průběžné výsledky mohou být až osmkrát větší.

Souřadnice bodů jsou popisovány dvojicí čísel. Souřadnice mohou být vzájemně sčítány a odčítány, dále násobeny a děleny číslem.

Cesta reprezentuje lomenou čáru nebo křivku, danou parametrickým vyjádřením.

Transformací může být libovolná kombinace otáčení, stejnolehlosti, zkosení a posunutí. Transformace bývá aplikována na cesty, obrázky a pera.

Datový typ `color` je podobný souřadnicovému typu; nejsou použity dvě komponenty, ale tři, přičemž jednotlivé komponenty (R, G, B) odpovídají postupně podílu červené, zelené a modré barvy. Velikost každé z komponent nesmí překročit meze intervalu $[0, 1]$. Předdefinovány jsou `black`, `white`, `red`, `green` a `blue` pro černou, bílou, červenou, zelenou a modrou barvu. Černá barva odpovídá $(0,0,0)$ a bílá $(1,1,1)$. Barvy se mohou navzájem sčítat a odčítat. Barvu lze také násobit reálným číslem. Je-li požadována šedá barva, například $(0.7,0.7,0.7)$, je možné použít zápis `0.7white`.

Řetězce jsou reprezentovány posloupností písmen v uvozovkách.

Booleovské proměnné nabývají hodnot `true` nebo `false` a existují pro ně operátory `and`, `or`, `not`.

Výsledky příkazů pro kreslení jsou ukládány do speciálních proměnných typu `picture`, například u příkazu `draw` je to `currentpicture`. Obrázky lze přidávat k jiným obrázkům a je možné na ně aplikovat afinní transformace.

Datovému typu `pen` je věnován odstavec Pera na straně 45.

Deklarace proměnných

K deklaraci proměnné se v METAPOSTu používá příkaz

```
typ název_proměnné .
```

Typ proměnné je jeden z výše uvedených datových typů. Pro deklaraci celého pole proměnných se použije `název_proměnné []`. Proměnné, kterým není přiřazen typ, jsou METAPOSTem chápány, jako by byly typu `numeric`. Je-li `název_proměnné` přiřazen nějakému typu, je tato deklarace platná v celém zdrojovém textu, ne pouze v jednotlivých obrázcích (zdrojový text totiž smí obsahovat popisy více obrázků, viz odstavec Vstupní soubor na straně 43).

Proměnné `zpřípona` jsou předdefinovány jako dvojice (`xpřípona`, `ypřípona`), přičemž `přípona` je složena z tokenů, o kterých se zmiňují v následujícím odstavci. Na začátku každého obrázku, to znamená vždy po příkazu `beginfig`, nejsou dvojicím `zpřípona`=(`xpřípona`, `ypřípona`) přiřazeny hodnoty.

Ke zjištění typu proměnné nebo její hodnoty slouží příkaz

```
show název_proměnné .
```

Tokeny

Tokeny jsou základními stavebními prvky METAPOSTu. Vstupní soubor se skládá z číselných tokenů, řetězových tokenů a symbolických tokenů.

Nejpoužívanějšími symbolickými tokeny jsou velká a malá písmena anglické abecedy a znak podtržítka (`_`). Mezi významné tokeny patří znak procenta (`%`), který způsobí ignorování zbývajícího kódu na aktuálním řádku, a také znak tečky (`.`), záleží však na počtu teček vyskytujících se za sebou. Dvě a více teček dohromady tvoří symbolický token (například `..` nebo `...`), tečka stojící před a za číslicemi je součástí číselného tokenu. V případě, že jedna tečka není obklopena číslicemi, a tedy není částí čísla, je tato tečka ignorována. Tohoto lze využít pro přehledné pojmenování proměnných, například `m.a`, přičemž se tento název skládá ze dvou tokenů `m` a `a`. Znaky `,` `;` `()` – čárka, středník a kulaté závorky – jsou považovány za samostatný token, i když stojí těsně za sebou.

Způsob, jak METAPOST zachází s tokeny, a tabulku tokenů najdeme v knize METAFONTbook ([3, str. 50]) a v manuálu k METAPOSTu ([2, str.16]).

Symbolické tokeny rozdělujeme do dvou skupin. Symbolický token bez speciálního významu, například jméno námi nadefinované proměnné, se nazývá **tag**. Symbolický token, který nese název primitivního příkazu (viz strana 56) nebo byl definován pomocí `def` jako makro (viz odstavec Makra na straně 56), se nazývá **spark**.

1.2. Vstupní soubor

Vstupní soubor mívá obvykle příponu `.mp`. Soubor `obrazek.mp` může vypadat například takto:

```
prologues:=1;
u:=1cm;
pen tluste;
```

```
beginfig(1);
z1=(u,u);
z2=(3u,3u);
draw z1--z2;
pickup pencircle scaled 3pt;
draw z1;
draw z2;
endfig;
```

```
beginfig(3);
draw fullcircle scaled 2u shifted (5u,5u) withcolor 0.3white;
tluste:=makepen(fullcircle scaled 0.3u);
```

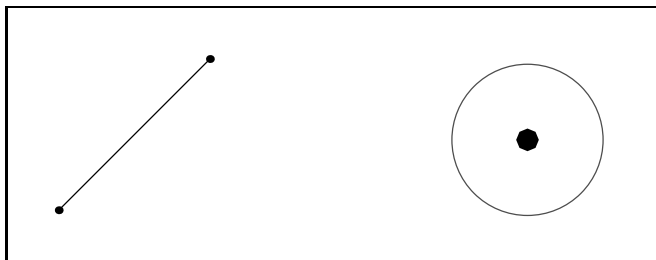
```
draw (5u,5u) withpen tluste;
endfig;
```

```
end
```

Význam prvního řádku obsahujícího přiřazení proměnné `prologues` vysvětluje odstavec Prologues (strana 62). Dvojice příkazů `beginfig(číslo) ... endfig` ohraničuje příkazy popisující jeden obrázek. V souboru může být popsáno obrázků několik. Má-li více obrázků stejné *číslo*, bude výsledný obrázek s daným *číslem* odpovídat popisu v pořadí posledního obrázku, který se ve vstupním souboru objevuje v okolí `beginfig(číslo) ... endfig`. Vstupní soubor ukončuje příkaz `end`.

Při spuštění překladače je možné u jména souboru vynechat příponu `.mp`. Pro překlad souboru `obrazek.mp` tedy postačí zadat `mpost obrazek`.

Uvedeným postupem ze vstupního souboru `obrazek.mp` vzniknou tři soubory, a to `obrazek.1`, `obrazek.3`, které obsahují program v jazyce PostScript vykreslující dané obrázky, a soubor `obrazek.log`, v němž je zaznamenán postup překladu a případná chybová hlášení.



Obrázek 1: obrazek.1 a obrazek.3

Obrázek si prohlédneme například pomocí programu Ghostview, nebo jej vložíme do $\text{T}_{\text{E}}\text{X}$ ovského dokumentu; ten zpracujeme odpovídajícím způsobem a prohlédneme vhodným prohlížečem. Vkládání obrázku je závislé na použitém formátu $\text{T}_{\text{E}}\text{X}$ u:

- v plain $\text{T}_{\text{E}}\text{X}$ u, $\mathcal{A}\mathcal{M}\mathcal{S}\text{-T}_{\text{E}}\text{X}$ u a $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ u 2.09: `\epsfbox {jméno_souboru}` (potřebujeme soubor `epsf.tex`),
- v $\text{pdfT}_{\text{E}}\text{X}$ u: `\convertMPtoPDF {jméno_souboru}{1}{1}` (jsou nezbytné soubory `supp-pdf.tex` a `supp-mis.tex`),
- v $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X } 2_{\epsilon}$ a $\text{pdfL}^{\text{A}}\text{T}_{\text{E}}\text{X}$ u: `\includegraphics {jméno_souboru}` (předpokládá načtení balíků `graphics` nebo `graphicx`, pro $\text{pdfL}^{\text{A}}\text{T}_{\text{E}}\text{X}$ s volbou `pdftex`, přičemž jsou ještě potřebné soubory `pdftex.def`, `supp-pdf.tex` a `supp-mis.tex`).

Protože METAPOST vytváří soubory s příponou *.číslo*, je při použití pdfL^AT_EXu nutno uvést příkaz

```
\DeclareGraphicsRule {*} {mps} {*} {} .
```

Po překladu dostaneme soubory s příponou *.dvi*, respektive *.pdf*. První z nich ještě zpracujeme programem *dvips* (neboť při prohlížení *.dvi* souboru nemusí být viditelné obrázky) a vzniklý postscriptový soubor prohlédneme pomocí Ghostview; soubor *.pdf* prohlédneme například prohlížečem Acrobat Reader.

1.3. Základní příkazy kreslení

Jednotky

METAPOST užívá základní systém souřadnic shodný se souřadným systémem jazyka PostScript. Jednotkou tohoto systému je postscriptový bod o velikosti $\frac{1}{72}$ palce, bývá označován **bp**. Dalšími jednotkami, které METAPOST zná, jsou **pt** o velikosti $\frac{1}{72,27}$ palce, **in** pro palec, **cm** pro centimetr a **mm** pro milimetr.

Pro jednodušší úpravu měřítka je výhodné přiřazení jednotky nějaké proměnné, *u:=cm* apod. Rozlišujeme *u=cm* (rovnice) a *u:=cm* (přiřazení). Následně *u=2cm* by dalo chybu, zatímco *u:=2cm* by změnilo hodnotu *u*.

Body

Ve všech příkazech kreslení je možno používat přímo souřadnicový zápis bodů, ale pohodlnější a přehlednější je nadefinování bodů v úvodu vstupního souboru, například *z1=(3cm,1cm)*. Bod *zčíslo* reprezentuje bod o souřadnicích (*xčíslo,yčíslo*). Souřadnice bodů je možné navzájem sčítat a odčítat. Souřadnice bodu lze násobit a dělit číslem. METAPOST také řeší lineární rovnice. Je tedy možné zadat body $Z_1 = [3, 1]$ a $Z_2 = [-3, 7]$ například takto:

```
x1=-x2=3cm;      x1+y1=x2+y2=4cm;
```

Střed úsečky lze zjistit jednoduše, pomocí zápisu $z3=1/2[z1,z2]$.

Jako pomocnou neznámou, jejíž přesná hodnota pro nás v danou chvíli není důležitá, používáme *whatever*. Velice vhodné je její použití při hledání průsečíku dvou přímek, tj. řešení soustavy lineárních rovnic:

```
z5=whatever*[z1,z2]=whatever*[z3,z4];
```

Hledaným průsečíkem je zde *z5*. Neznámou *whatever* lze použít vícekrát, jednotlivé hodnoty jsou na sobě nezávislé.

Pera

Jedním ze základních požadavků pro kreslení je umožnění změny tloušťky pera. Příkazem

```
pencircle scaled číslo
```

zvolíme kruhové pero zadané tloušťky. Na začátku každého obrázku, tedy po příkazu `beginfig`, je nastaveno pero o tloušťce 0,5 bp. Je-li třeba získat například pero s kaligrafickým efektem, může být použita některá z lineárních transformací. Vlastní nastavení požadovaného pera se provede příkazem

```
pickup název_pera .
```

K opětovnému získání přednastavené hodnoty tloušťky pera lze použít příkazu

```
pickup defaultpen .
```

METAPOST disponuje také perem `pensquare`, jež je vytvořeno příkazem

```
makepen((- .5, -.5)--(.5, -.5)--(.5, .5)--(-.5, .5)--cycle) .
```

Je tedy zřejmé, že příkaz `makepen` *cesta* vytvoří pero tvaru dané cesty. Opačným příkazem je `makepath` *pero*, který danému peru přiřadí jako výstup odpovídající cestu.

Body, lomené čáry a křivky

Bod se nakreslí příkazem

```
drawdot bod .
```

Lomená čára se vykreslí příkazem

```
draw bod--bod--bod ,
```

uzavřená lomená čára potom

```
draw bod--bod--bod--cycle .
```

METAPOST kreslí Bézierovy kubiky procházející zadanými body. Sám si vypočítá kontrolní body tak, aby se křivka jevila co „nejhezčí“. Příkaz

```
draw bod..bod..bod
```

popisuje Bézierovu křivku a příkaz

```
draw bod..bod..bod..cycle
```

udává uzavřenou křivku.

Dále máme několik možností, jak tvar křivky upravit podle svých představ.

Sklon tečny v bodě křivky ovlivníme příkazem

```
draw bod..bod{dir číslo}..dir číslo}bod .
```

Můžeme také použít předdefinované směry `up`, `down`, `left`, `right`.

Napětí křivky se upraví příkazem

```
draw bod..bod..bod..tension číslo and číslo..bod..bod .
```

Číslo je desetinné číslo větší než $\frac{3}{4}$; hodnota 1 odpovídá použití „..“. Pro `tension` nekonečno je kromě `tension infinity` k dispozici „...“.

Zakřivení cesty lze ovlivnit příkazem

```
draw bod{curl číslo}..bod..curl číslo}..bod .
```

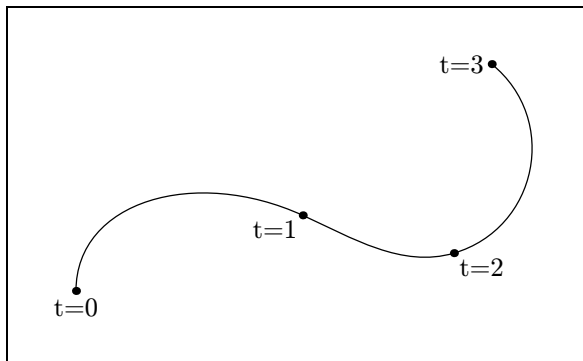
Číslo menší než 1 snižuje zakřivení, *číslo* větší než 1 jej zmenšuje. Poslední možností je přímé zadání kontrolních bodů příkazem

```
..controls kontrolní_bod and kontrolní_bod.. .
```

Pro vymazání slouží příkazy `undrawdot` a `undraw` se stejnou syntaxí.

Parametrizace křivek

METAPOST pracuje s cestami jako parametrizovanými křivkami a tím umožňuje zjišťovat o cestách (křivkách) mnoho užitečných informací. Křivka definovaná parametricky je zapisována jako množina bodů $(X(t), Y(t))$, kde t se pohybuje od nuly k číslu udávajícímu počet křivkových segmentů (tj. k číslu o jedničku menšímu než počet zadaných bodů křivky). Hodnota t se nazývá čas.



Obrázek 2: Křivka definovaná parametricky

Průsečík dvou křivek se zjišťuje příkazy

```
a intersectionpoint b ,  
a intersectiontimes b .
```

kde a a b jsou cesty. Výsledkem prvního příkazu jsou souřadnice průsečíku (neprotínají-li se zadané cesty, METAPOST hlásí chybu); výsledkem druhého je dvojice (t_a, t_b) , t_a je čas na cestě a v průsečíku cest a a b , t_b je odpovídající čas na cestě b . Je-li výsledkem $(-1, -1)$, cesty se neprotínají. Může se stát, že se dvě křivky protínají ve více bodech; METAPOST v tomto případě vybírá (t_a, t_b) s nejmenším t_a , to je ovšem velmi zjednodušeně řečeno, ve složitějších případech jsou prováděna ještě další porovnávání. Podrobnější vysvětlení najdeme v knize METAFONTbook ([3]).

Pro zjištění souřadnice bodu na křivce slouží příkaz

```
point t of cesta .
```

Čas celé křivky udává příkaz

```
length cesta .
```

Příkaz

```
subpath (t1, t2) of cesta
```

vyjme z cesty úsek mezi časy t_1 a t_2 .

S operací `subpath` pracují další dva příkazy

```
a cutbefore b ,
```

`a cutafter b` ,

kde a a b jsou cesty, jejichž průsečík je bod P . První z příkazů vybere úsek cesty a od bodu P ke konci, druhý vybere naopak začátek a až k bodu P .

Tečný vektor křivky v jejím libovolném t vrací příkaz

`direction t of cesta` ,

t příslušný zadanému tečnému vektoru zase příkaz

`directiontime vektor of cesta`

a bod odpovídající tečnému vektoru příkaz

`directionpoint vektor of cesta` .

Příkazem

`arclength cesta`

se zjišťuje oblouková míra křivky.

Pro zjištění času odpovídajícího obloukové míře a na křivce p je připraven příkaz

`arctime a of p` .

Afinní transformace

METAPOST disponuje následujícími transformacemi:

- posunutí o vektor (a, b) : (x, y) `shifted (a, b)` $\longrightarrow (x + a, y + b)$,
- otočení o úhel θ , který je zadaný ve stupních, kolem středu $(0, 0)$ proti směru hodinových ručiček: (x, y) `rotated θ` $\longrightarrow (x \cos \theta - y \sin \theta, x \sin \theta + y \cos \theta)$,
- zkosení s koeficientem a : (x, y) `slanted a` $\longrightarrow (x + ay, y)$,
- stejnolehlost s koeficientem a : (x, y) `scaled a` $\longrightarrow (ax, ay)$,
- změna měřítka na ose x : (x, y) `xscaled a` $\longrightarrow (ax, y)$,
- změna měřítka na ose y : (x, y) `yscaled a` $\longrightarrow (x, ay)$,
- rotace a stejnolehlost: (x, y) `zscaled (a, b)` $\longrightarrow (ax - by, bx + ay)$
(což odpovídá násobení komplexních čísel),
- osová souměrnost podle přímky procházející body a a b :
 (x, y) `reflectedabout (a, b)`,
- otočení o úhel θ , který je zadaný ve stupních, kolem středu (a, b) proti směru hodinových ručiček: (x, y) `rotatedaround ((a, b), θ)`.

Použití transformací ukazuje obrázek 3.

```
u=.7cm;
```

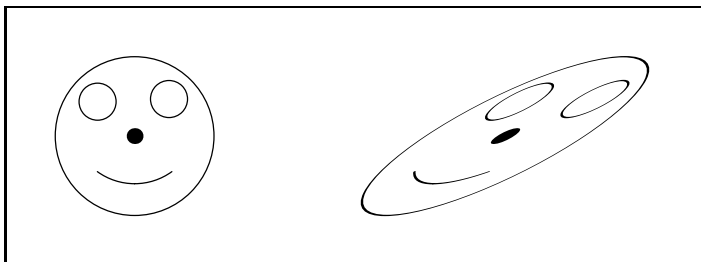
```
beginfig(1);
```

```
path p[]; transform t[]; picture obrazek[];
```

```
p1 = fullcircle scaled 3u;
```

```
p2 = subpath (5.2,6) of p1;
```

```
t1 = identity scaled .8 shifted (0,.3u);
```



Obrázek 3: Použití afinních transformací

```

p3 = p2 transformed t1;

p4 = fullcircle scaled .7u shifted (.65u,.7u);
t2 = identity rotated 90;
p5 = p4 transformed t2;

t3 = identity reflectedabout ((0,-2u),(0,2u));
p6 = p3 transformed t3;

pickup pencircle scaled .3u;
draw (0,0);
pickup defaultpen;

draw p1; draw p3; draw p4; draw p5; draw p6;

obrazek1:=currentpicture;
t4 = identity slanted 1.5 shifted (7u,0);
obrazek2 = obrazek1 transformed t4;
draw obrazek2;
endfig; end

```

Chceme-li k zadané transformaci t provést transformaci inverzní, použijeme příkaz:

$$p = q \text{ transformed inverse } t \quad .$$

Kružnice

Pro kreslení kružnice je předdefinována cesta `fullcircle`, která popisuje kružnici o jednotkovém průměru a středu v počátku. Dále nám METAPOST nabízí cestu `halfcircle`, což je část kružnice se středem v počátku a jednotkovým průměrem nad osou x , a `quartercircle` odpovídající části kružnice o jednotkovém průměru se středem v počátku ležící v prvním kvadrantu.

Kružnice má parametrickou délku 8.

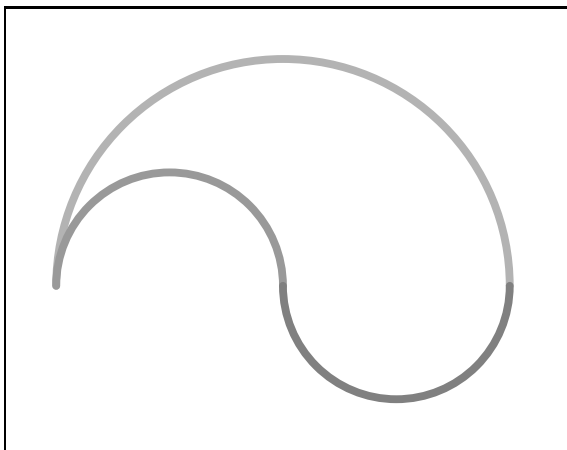
```
u:=cm;
path p[];

p1 = halfcircle scaled 6u;
p2 = halfcircle scaled 3u shifted (-1.5u,0);
p3 = p2 rotated 180;

beginfig(1);

pickup pencircle scaled 3pt;
draw p1 withcolor 0.7white;
draw p2 withcolor 0.6white;
draw p3 withcolor 0.5white;

endfig;
end
```



Obrázek 4: Obrázek odpovídající předchozímu zdrojovému kódu

Přerušované čáry, krajní body čar a šipky

METAPOST poskytuje kromě plné čáry i čáry přerušované, a to tečkované a čárkované. Pro jejich vykreslení používáme příkaz

```
draw cesta dashed vzorek ,
```

kde *vzorek* je typu *picture*. Předdefinovány jsou *vzorky* *evenly* a *withdots*.

První z nich vykresluje čárkovanou čáru tvořenou čárkami délky 3 pt a mezerami o velikosti 3 pt, druhý kreslí tečkovanou čáru s tečkami vzdálenými od sebe 5 pt.

Vzhled přerušované čáry si můžeme přizpůsobit svému přání jedním z následujících způsobů:

- prodloužením čárek nebo mezer mezi tečkami – transformace **scaled**
- posunutím čárek, respektive teček – transformace **shifted**
- nadefinováním mezer a délky čárek – příkaz **dashpattern** (*zápis on|off*)

Vnitřní proměnné **linecap**, ovlivňující tvar čáry v jejích krajních bodech, můžeme přiřadit jedno ze tří možných nastavení – **rounded** pro zaoblení čáry v krajních bodech, **butt** pro „rovné“ zakončení čáry a **squared**. V případě **butt** je čára dlouhá přesně tak, jak je zadána, v případě **rounded** a **squared** se délka čáry prodlouží na obou koncích o tloušťku pera.

Jiná vnitřní proměnná, **linejoin**, ovlivňuje rohy při změně směru lomené čáry. Může nabývat hodnot **rounded**, **beveled**, **mitered**.

Chceme-li čáru zakončit šipkou, použijeme příkaz

```
drawarrow cesta
```

na místě příkazu **draw cesta**. Bude vykreslena zadaná cesta se šipkou v posledním bodu křivky. Pro šipku na začátku křivky využijeme příkaz

```
drawarrow reverse cesta
```

a šipky na obou koncích křivky vykreslíme pomocí příkazu

```
drawdblarrow cesta
```

Velikost šipky ovlivňuje hodnota vnitřní proměnné **ahlength**; úhel, který svírá šipka s křivkou, vyjadřuje proměnná **ahangle**.

Podrobnosti a ukázky všech předešlých proměnných a příkazů v odstavci najdeme v manuálu k METAPOSTu ([2, str. 32]).

1.4. Vkládání textu a obrázků

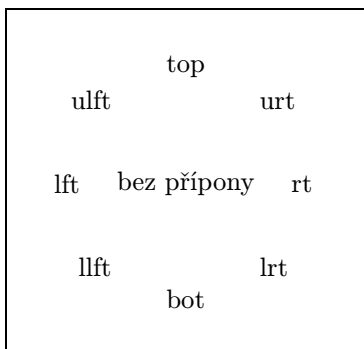
Nejjednodušší způsob, jak v METAPOSTu vložit do obrázku text nebo obrázek, je použitím příkazu

```
label.přípona_pro_umístění(řetězec_nebo_obrázek, souřadnice) ,
```

kde *řetězec* představuje text v uvozovkách. Následující obrázek ukazuje polohu textu nebo obrázku odpovídající jednotlivým *příponám_pro_umístění*.

Kód tohoto obrázku vypadá takto (je uveden bez **beginfig**, **endfig** a **end**):

```
defaultfont="csr10"; u=.7cm; labeloffset:=2u; z1=(0,0);  
verbatimtex \font\muj=csr10 \muj etex;  
label(btex bez přípony etex,z1);  
label.rt("rt",z1); label.urt("urt",z1);  
label.lft("lft",z1); label.ulft("ulft",z1);  
label.top("top",z1); label.llft("llft",z1);  
label.bot("bot",z1); label.lrt("lrt",z1);
```



Obrázek 5: Přípony používané pro umísťování textu nebo obrázku

Vnitřní proměnná `labeloffset` určuje vzdálenost vkládaného textu od bodu `z1`. Přednastavená hodnota této proměnné je `3bp`.

Nahradíme-li `label` příkazem `dotlabel`, umístí se na zadané souřadnici tečka.

Další způsob, jak umístit text nebo obrázek, je pomocí příkazu `thelabel`, který má obdobnou syntaxi jako `label` a `dotlabel`. Výstupem příkazu `thelabel` je datový typ `picture`, což znamená, že `thelabel` přímo nevykreslí požadovaný text nebo obrázek.

Příkaz

```
label.bot("M", (0,0));
je tedy ekvivalentní příkazu
draw thelabel.bot("M", (0,0));
a také příkazu
picture a;
a = thelabel.bot("M", (0,0));
draw a;
```

Font textu můžeme změnit přiřazením

```
defaultfont := "název_fontu" .
```

Přednastavený je font `cmr10`. Pro kreslení našich obrázků zvolíme font `csr10`. V Linuxu pak můžeme použít české znaky, neboť používá systémové kódování ISO-8859-2, jež se shoduje s kódováním fontu `csr10`; v jiných operačních systémech (Windows, OS/2) jsou některé znaky s diakritikou chybné, protože systémové kódování se neshoduje s kódováním fontu `csr10`.

Další nevýhodou je, že tento font neobsahuje znak `space`, takže v řetězci nemohou být mezery. V tom případě jsou vhodnější například fonty `rphvr` (Helvetica) nebo `rptmr` (Times Roman), ale ty zase nemají všechny české znaky. Pokud potřebujeme obojí, musíme mít buď vhodný český postscriptový font

včetně T_EXovské metriky (například `phvr8z` či `ptmr8z`), nebo raději použijeme okolí `btex ... etex` (viz dále).

Vnitřní proměnné `defaultscale` je přiřazen koeficient stejnolehlosti velikosti fontu. Předdefinovaná hodnota `defaultscale` je 1, což většinou odpovídá velikosti 10 bodů. Jestliže neznáme základní velikost fontu a chceme-li zvětšit písmo například na 12pt, použijeme přiřazení:

```
defaultscale := 12pt/fontsize defaultfont;
```

Text v METAPOSTu

Pro vkládání náročnějšího textu můžeme využít příkazy T_EXu. Použijeme-li ve vstupním souboru METAPOSTu

```
btex příkazy_TEXu etex ,
```

jsou *příkazy_TEXu* vysázeny T_EXem a výsledek je předán zpět METAPOSTu jako datový typ `picture`, což nám umožňuje text otáčet a jinak přizpůsobovat našim požadavkům.

Překladač `mpost` v případě výskytu okolí `btex ... etex` ve vstupním souboru hledá soubor `makempx`. Adresáře, ve kterých `mpost` tento soubor hledá, jsou závislé na operačním systému. Například v operačním systému Linux `mpost` při hledání konzultuje obsah proměnné `PATH`. Pokud je v této proměnné i tečka (`.`), prohledává i aktuální adresář. V ostatních operačních systémech to může být jinak. Standardní postup popsany v `makempx`, jež je součástí instalace METAPOSTu, je zhruba následující:

- Nejprve je zkontrolováno, zda-li jsme po posledním překladu vstupní soubor `obrázek.mp` měnili – test programem `NEWER`. V případě, že ano, pokračuje se následujícími kroky; v opačném případě nejsou tyto kroky provedeny.
- Okolí `btex ... etex` jsou uložena do souboru, který se v operačním systému Linux jmenuje `mpx$$.tex` (`$$` je číslo procesu), v operačních systémech OS/2 a Windows je tento soubor nazván `mpx_tmp.tex` – program `MPTOTEX`. V Linuxu je navíc možnost přidání nějaké hlavičky (proměnná `$MPTEXPRE`).
- Na vytvořený soubor je spuštěn T_EX s parametry, které jsou v `makempx` předdefinovány a my si je zde můžeme změnit. Také je výhodné připsat příkaz, který způsobí výpis případných chybových hlášení přímo při překladu na obrazovku. Takto jsme získali soubor `mpx$$.dvi` (nebo `mpx_tmp.dvi`).
- Ze souboru `mpx$$.dvi` (`mpx_tmp.dvi`) je vytvořen soubor `obrázek.mpx`, který obsahuje kód v METAPOSTu, to jest popis obrázků – program `DVITOMP`. Na konci každého obrázku je příkaz `mpxbreak`. Nyní, vrátíme-li se k prvnímu bodu, můžeme říci, že je testováno, je-li soubor `obrázek.mpx` starší než `obrázek.mp`.
- Nakonec jsou smazány všechny dočasně vytvořené soubory.



Obrázek 6: Bounding box

Popisované kroky tedy lze ovlivňovat modifikací souboru `makempx` (v operačních systémech jiných než Linux s příponou označující spustitelný soubor).

Zjednodušeně si můžeme tento postup představit tak, že `mpost` si vybere všechna okolí `btex ... etex` do nějakého souboru, který si zpracuje do podoby pro něj přijatelné. A při svém druhém průchodu už má nachystáno vše potřebné, aby dokázal popisky vykreslit.

Do vstupního souboru `METAPOSTu` můžeme dále vložit různé definice a pomocné příkazy `TEXu` uzavřené v okolí `verbatimtex ... etex`. Hlavním rozdílem je, že `btex` vytváří obrázek, zatímco `verbatimtex` pouze vkládá příkazy `TEXu`.

Do okolí `verbatimtex ... etex` smíme zapsat např. celou hlavičku kódu `LATEXu` až po `\begin{document}` (včetně). Pokud v `makempx` specifikujeme použití `LATEXu` místo přednastavených parametrů `TEXu`, budeme text v okolí `btex ... etex` zapisovat v kódu `LATEXu`.

Měření textu

Předdefinované makro

```
bbox proměnná ,
```

kde *proměnná* je typu `picture`, `path` nebo `pen`, reprezentuje bounding box obrázku (respektive cesty nebo pera). Bounding boxem obrázku rozumíme nejmenší obdélník obsahující tento obrázek. Je-li `p` proměnná typu `picture`, je příkaz `draw bbox p;`

ekvivalentní příkazu

```
draw llcorner p-(bboxmargin,bboxmargin)
  -- lrcorner p+(bboxmargin,-bboxmargin)
  -- urcorner p+(bboxmargin,bboxmargin)
  -- ulcorner p+(-bboxmargin,bboxmargin)
  -- cycle;
```

Vnitřní proměnná `bboxmargin` má přednastavenou hodnotu 2 bp.

1.5. Výplně

Vyplňovat smíme v METAPOSTu pouze uzavřené křivky, a to křivky uzavřené příkazem `..cycle` nebo `--cycle`. Syntaxe příkazu pro vybarvení je:

```
fill cesta withcolor barva .
```

Pokud neudáme žádnou barvu, METAPOST vybarví plochu černě.

Chceme-li plochu naopak „odbarvit“, použijeme příkaz `unfill`, který je definován jako `fill cesta withcolor background` (proměnná `background` je v souboru `plain.mp` předdefinována jako bílá barva).

Kombinací vyplňování a kreslení je příkaz

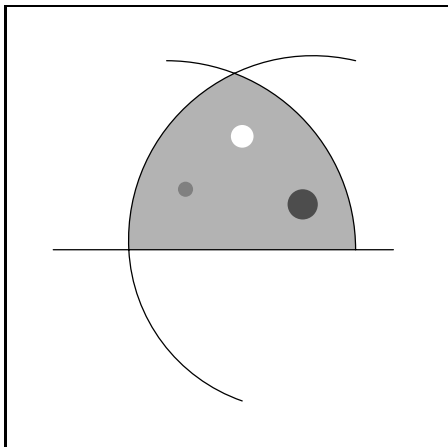
```
filldraw cesta withcolor barva ,
```

respektive příkaz `unfilldraw cesta` pro odbarvování.

Příkaz

```
buildcycle (p1, p2, ..., pk)
```

vybere průsečíky křivek p_k, p_1 ; p_1, p_2 ; ...; p_{k-1}, p_k a vytvoří uzavřenou cestu reprezentující hranici oblasti ležící „mezi“ všemi zadanými křivkami. Průsečík křivek p_i a p_{i+1} je vyhledáván tak, aby byl co nejpozdějším na p_i a co nejdřívejším na p_{i+1} .



Obrázek 7: Ukázka vyplňování pomocí příkazu `buildcycle`

Zdrojový kód k obrázku 7:

```
u=cm;          path p[];

p1 = quartercircle scaled 5u;
p2 = (-1.5u,0) -- (3u,0);
p3 = (u,-2u) .. (-.5u,0) .. (2.5u,2.5u);
```

```

p4 = buildcycle (p1,p2,p3);

p5 = fullcircle scaled .3u shifted (u,1.5u);
p6 = fullcircle scaled .2u shifted (.25u,.8u);
p7 = fullcircle scaled .4u shifted (1.8u,.6u);

fill p4 withcolor .7white;
draw p1; draw p2; draw p3;
unfill p5;
background:=.5white;%%% rovnocennými příkazy jsou:
unfill p6;          %%% fill p6 withcolor .5 white;
background:=.3white;
unfill p7;          %%% fill p7 withcolor .3 white;

```

1.6. Makra

Některé příkazy zmíněné v tomto textu nejsou přímo „vestavěny“ v METAPOSTu („vestavěné“ příkazy nazýváme primitivní). METAPOST automaticky využívá soubor `plain.mp`, ve kterém jsou uloženy definice mnoha příkazů ulehčujících nám kreslení. Cílem tohoto odstavce je objasnění zápisu podobných maker.

Makro

```
def posloupnost_tokenů = nahrazující_text enddef;
```

umožňuje použití *posloupnosti_tokenů* místo *nahrazujícího_textu*.

Makro s parametry

```
def kruz (expr s, r) =
  draw fullcircle scaled r shifted s;
enddef;
```

se liší pouze tím, že ukazuje, kde se mají v těle definice použít argumenty *s* a *r*, přičemž zde to mohou být libovolné výrazy.

Příkaz `def` lze nahradit příkazem `vardef`. Toto makro má podobnou syntaxi jako makra definovaná pomocí `def`, jen v úvodu na místě `def posloupnost_tokenů` stojí `vardef všeobecná_proměnná`, kde *všeobecná_proměnná* je jméno proměnné, jehož číselná část je nahrazena obecným symbolem pole `[]`. Jméno následující po `vardef` je tímto nedefinováno stejně jako při deklaraci proměnné. Hlavní rozdíl mezi makrem definovaným `def` a makrem definovaným `vardef` je ten, že makro definované `vardef` má automaticky vloženou skupinu `beginingroup... endgroup` na začátku a na konci *nahrazujícího_textu* (viz níže odstavec Seskupování). Podrobnější vysvětlení práce s makry definovanými `vardef` a dalšími typy definic můžeme najít v manuálu k METAPOSTu ([2, str. 49]).

V makrech METAPOSTu můžeme využít lokálních proměnných, cyklů i podmíněných příkazů.

Seskupování

Skupinou nazýváme posloupnost příkazů, oddělených středníky, ohraničenou příkazy `begingroup ... endgroup`.

Toho, aby byla proměnná, například proměnná m , lokální, dosáhneme příkazem `save m`. Všechny proměnné, jejichž název začíná m (například tedy m , $m.m$, $m.7$), se tímto stanou proměnnými typu `numeric` a jejich dřívější hodnoty jsou „zapomenuty“ až do uzavření skupiny. Použití `save p` mimo skupinu způsobí úplné zničení předcházejících hodnot proměnné. Takto se tedy budou měnit hodnoty proměnné p :

```
p=7;           % - hodnota p je 7
begingroup;    % - začátek skupiny
show p;        % - hodnota p >>7
save p;        % - uschování hodnoty p
show p;        % - hodnota p >>p
endgroup;      % - konec skupiny
show p;        % - hodnota p >>7
save p;        % - uschování hodnoty p;
               %   úplné zničení hodnoty p
show p;        % - hodnota p >>p
```

Důležité pro nás je, že příkaz `save` umožňuje použití stejně pojmenovaných proměnných uvnitř skupiny i mimo ni a opakované volání jednoho makra.

Složitější makra

Makro vykreslující kružnici o daném poloměru r a středu s můžeme zapsat například takto:

```
def k~(expr r, s) =
  draw fullcircle scaled (2*r) shifted s;
enddef;
```

Chceme-li nakreslit kružnici o poloměru 3 a středu [3, 4], použijeme nadefinované makro pomocí příkazu `k(3u, (3u, 4u))`.

V makrech lze využít podmíněný příkaz se syntaxí

```
if podmínka: posloupnost příkazů else: posloupnost příkazů fi .
```

Vkládáme-li podmíněný příkaz do jiného podmíněného příkazu, použijeme zkrácený zápis

```
if 1. podmínka: ... elseif 2. podmínka: ... else: ... fi .
```

Zvláště ve složitějších makrech oceníme skutečnost, že *posloupnost příkazů* nemusí být ukončeným příkazem. Například:

```
def kruz (expr r, s) =
  draw fullcircle scaled
    if r > 5cm: r else: (2*r) fi
```

```
    shifted s;  
enddef;
```

Parametry

V předcházejících makrech jsme používali pouze parametry typu `expr`, což mohly být výrazy libovolného datového typu. Kromě `expr` jsou k dispozici parametry typu `suffix` a `text`, které deklarují libovolné jméno nebo libovolnou posloupnost tokenů. Jako ukázkou makra s parametrem typu `text` vytvoříme makro `ramecek`.

```
def ramecek (text t) =  
    draw t;  
    draw bbox t;  
enddef;
```

vykreslující t a rámeček kolem t .

Přidáme parametr typu `expr` pro barvu rámečku:

```
def ramecek (text t) (expr barva) =  
    draw t;  
    draw bbox t withcolor barva;  
enddef;
```

Makro zavoláme například příkazem
`ramecek(fullcircle scaled 5u)(green)`.

Makro `incr` využívá parametr typu `suffix`:

```
def incr (suffix $) =  
    $:=$+1;  
enddef;
```

Toto makro zvýší hodnotu numerické proměnné o jedničku. Všimněme si, že parametr typu `suffix` se vyskytuje na levé i pravé straně přiřazovacího příkazu (`:=`), což s jiným typem parametru nelze.

Makro, ve kterém figurují parametry typu `suffix` a `expr` společně, například

```
def moje (suffix a) (expr b) = label(str a,b); enddef;  
voláme moje(ahoj, (3cm,3cm)) nebo moje(ahoj)((3cm,3cm)). Příkaz str převede parametr typu suffix na řetězec.
```

Cykly

Při kreslení obrázků či vytváření maker určitě využijeme cyklus `for`. Můžeme rozlišit čtyři základní druhy cyklu `for`. První z nich zapisujeme

```
for posloupnost_tokenů = výraz1, výraz2, ..., výrazn:  
    posloupnost_příkazů  
endfor .
```

Cyklus provede *posloupnost_příkazů* postupně s hodnotami *posloupnosti_tokenů* rovnajícími se *výrazu_k*, kde $k = 1, 2, \dots, n$. *Výraz_k*, $k = 1, 2, \dots, n$ nemusí mít v danou chvíli přiřazenu konkrétní hodnotu.

Obecná syntaxe druhého a nejobecnějšího cyklu typu `for` je:

```
for posloupnost_tokenů = výraz1 step krok until výraz2:
    posloupnost_příkazů
endfor .
```

Cyklus nejprve vyhodnotí, zda jsou *výrazu₁*, *výrazu₂* a *kroku* přiřazeny nějaké konkrétní numerické hodnoty. Dále postupuje takto:

Je-li *krok* > 0 a *výraz₁* > *výraz₂*, nebo *krok* < 0 a *výraz₁* < *výraz₂*, *posloupnost_příkazů* není provedena. V případě, že ani jedna z předcházejících podmínek splněna není, je provedena *posloupnost_příkazů* pro hodnotu *výrazu₁*. *Výraz₁* je nahrazen hodnotou (*výraz₁*+*krok*). Tento proces se opakuje s novou hodnotou *výrazu₁*.

Místo `step 1 until` je možné použít předdefinovaného příkazu `upto`. Příkaz `step -1 until` může být nahrazen příkazem `downto`.

Pro hodnoty *kroků* je doporučeno používat celá čísla nebo přesné násobky hodnoty zlomku $\frac{1}{65536}$. Jinak nemusí *posloupnost_tokenů* vůbec dosáhnout požadované konečné hodnoty. Například hodnoty *i* v průběhu cyklu

```
for i=0 step 0.1 until 1: show i; endfor;
```

jsou tyto:

```
> 0
> 0.1
> 0.20001
> 0.30002
> 0.40002
> 0.50003
> 0.60004
> 0.70004
> 0.80005
> 0.90005
```

Abychom se vyvarovali podobných případů, použijeme *krok*, který je celým číslem, a vhodně upravíme *posloupnost_příkazů* vynásobením či vydělením odpovídající proměnné k získání požadovaných hodnot. Například pro

```
for i=0 upto 10: show i/10; endfor;
```

dostaneme již výstup odpovídající naší představě:

```
> 0
> 0.1
> 0.2
> 0.3
> 0.4
> 0.5
> 0.6
> 0.7
> 0.8
```

```
> 0.9
> 1
```

Podobně jako u podmíněných příkazů smí být *posloupnost_příkazů* nejen jeden příkaz nebo více příkazů oddělených středníky, ale i příkaz neukončený, například:

```
draw for a=(0,0),(u,7u),(3u,5u): a~-- endfor cycle;
```

Tento příkaz je ekvivalentní příkazu:

```
draw (0,0)--(u,7u)--(3u,5u)--cycle;
```

Index *i* v cyklu *for* odpovídá parametru typu *expr*, může nabývat jakékoli hodnoty, ale není to proměnná, a tedy nemůžeme její hodnotu změnit přiřazovacím příkazem. To nám umožňuje cyklus *forsuffixes*, třetí druh cyklu *for*, jehož index se chová jako parametr typu *suffix*. Syntaxi a podrobnosti použití cyklu *forsuffixes* lze najít v manuálu k METAPOSTu ([2, str. 53]).

Cyklus

```
forever: posloupnost_příkazů endfor
```

provádí nekonečnou smyčku a je posledním z cyklů *for*.

Pro ukončení cyklu v okamžiku, kdy booleovská proměnná dosáhne pravdivé hodnoty, se používá příkaz

```
exitif booleovská_proměnná .
```

Potřebujeme-li ukončit cyklus po dosažení hodnoty nepravdivé, využijeme příkazu

```
exitunless booleovská_proměnná .
```

Propojením nekonečné smyčky a *exitunless* vznikne obdoba cyklu *while* známého z jiných programovacích jazyků:

```
forever: exitunless booleovská_proměnná; posloupnost_příkazů
endfor .
```

Užitečné makro

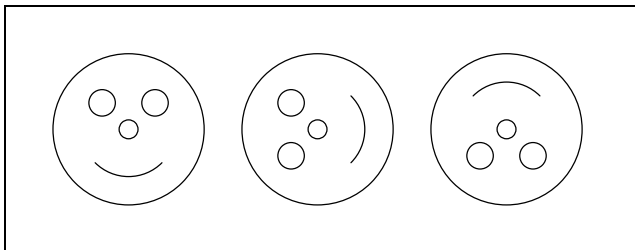
Pěkným a užitečným makrem je makro *image*, které najdeme v již zmíněném souboru *plain.mp*:

```
vardef image(text t) =
  save currentpicture;
  picture currentpicture;
  currentpicture := nullpicture;
  t;
  currentpicture
enddef;
```

Novým prvkem, který toto makro přináší, je předposlední řádek. Je-li před *enddef* proměnná, do níž je v makru něco přiřazováno, při volání makra použijeme:

```
proměnná = název_makra (parametry makra) ,
```

příčemž *proměnná* musí být stejného typu jako proměnná v makru na zmíněném řádku. Potom máme v *proměnné* uložen výsledek makra a můžeme ho vykreslit, transformovat a podobně.



Obrázek 8: Obrázek odpovídající následujícímu zdrojovému textu

```

u:=.5cm;

beginfig(1);

picture obrazek[];
path a[];
transform t[];
a1 = fullcircle scaled 4u;
a2 = fullcircle scaled 0.5u;
a3 = fullcircle scaled 0.7u shifted (0.7u,0.7u);
a4 = fullcircle scaled 0.7u shifted (-0.7u,0.7u);
a5 = subpath (5,7) of fullcircle scaled 2.5u;

obrazek1 = image(
    for i=1 upto 5:
        draw a[i];
    endfor;
);

t1 = identity rotated 90 shifted (5u,0);
t2 = identity rotated 180 shifted (10u,0);
obrazek2 = obrazek1 transformed t1;
obrazek3 = obrazek1 transformed t2;
for i=1 upto 3:
    draw obrazek[i];
endfor;

```

```
endfig;  
end
```

1.7. Prologues

Prohlížíme-li přímé výstupy METAPOSTu, je vhodné v úvodu každého vstupního souboru přiřadit speciální proměnné `prologues` kladnou hodnotu. V souboru `obrazek.mp` na straně 43 bylo použito `prologues:=1`. Tento příkaz způsobí, že v hlavičce souboru `obrazek.1` (a samozřejmě i v hlavičce souboru `obrazek.3`) se objeví

```
%!PS-Adobe-3.0 EPSF-3.0
```

Význam tohoto řádku je zhruba následující: vznikne soubor ve formátu EPS (Encapsulated PostScript), tzv. zapouzdřený PostScript. Formát EPS byl vytvořen kvůli možnosti vkládání těchto obrázků do jiných dokumentů.

Vkládáme-li do souboru text (obyčejný text, bez nějakých zvláštních znaků, například bez znaků matematických), to jest používáme-li nějaký font, je proměnná `prologues` opět důležitá. Při jejím správném nastavení postscriptový soubor obsahuje:

```
%%DocumentFonts: CMR10
```

```
 /cmr10 /CMR10 def
```

```
 /fshow {exch findfont exch scalefont setfont show}bind def
```

Toto by mělo způsobit vyhledání námi požadovaného fontu. Ovšem při prohlížení pomocí programů, které používají k vyrastování obrázků `Ghostscript` (například `Ghostview`), je font `cmr10` nahrazen fontem `Courier` díky standardnímu obsahu souboru `Fontmap` (pokud jej na příslušném místě neupravíme).

Problémy nastanou u složitějšího textu, neboť mapa fontů se může lišit a výsledek nemusí odpovídat naší představě. V takovém případě je lepší obrázek vložit do $\text{T}_{\text{E}}\text{X}$ ovského dokumentu a soubor `.dvi` vzniklý z tohoto dokumentu zpracovat programem `dvips` (přitom proměnnou `prologues` nenastavujeme). Řešením této situace je také využití programu `dvips` s volbou `j0` (spouštíme `dvips -j0`) při vytváření postscriptového souboru (nezáleží na tom, zda je proměnná `prologues` nastavena, či nikoli).

Přednastavená hodnota `prologues` je rovna 0.

Pro nás znamená příkaz `prologues:=c`, $c \leq 0$ to, že obrázek není oříznut (při $c > 0$ je oříznut na nejmenší možný obdélník), což nám při vkládání do textu nevádí, a při pokusu o prohlédnutí obrázku (myšleno výsledku překladu METAPOSTu) s popiskem dostáváme chybové hlášení o nenalezení fontu.

1.8. METAFONT a METAPOST

Hlavním rozdílem mezi METAFONTem a METAPOSTem je jejich rozdílný výstup. Výsledkem práce METAFONTu je bitová mapa a metrika, zatímco výstupem METAPOSTu je program v jazyce PostScript.

V manuálu k METAPOSTu ([2, str. 79]) jsou vypsány příkazy a proměnné nacházející se pouze v METAPOSTu a také ty, které najdeme jen v METAFONTu. Pro METAPOST jsou jedinečné příkazy týkající se barev, vkládání textu, obloukové délky křivky; METAFONT zase poskytuje proměnné a příkazy pro práci s písmeny (ligatury, kerningové páry), pixely a podobně.

Literatura

- [1] Adobe Systems Incorporated *PostScript Language Reference Manual*. Massachusetts: Addison-Wesley, 3. vydání, 1999. ISBN 0-201-37922-8
<http://adobe.com:80/products/postscript/pdfs/PLRM.pdf>
- [2] Hobby, J. D. *A User's Manual for MetaPost*. Součást dokumentace programu METAPOST.
<http://cm.bell-labs.com/cm/cs/cstr/162.ps.gz>
- [3] Knuth, D. E. *The METAFONTbook*. Massachusetts: Addison-Wesley, 1986. ISBN 0-201-13445-4
- [4] Kuben, J. *Diferenciální počet funkcí jedné proměnné*. Brno: VA, 1999
- [5] Kuben, J. *Zpravodaj Československého sdružení uživatelů T_EXu 2/94*. Brno.
<http://bulletin.cstug.cz/pdf/bul942.pdf>
- [6] Olšák, P. *T_EXbook naruby*. Brno: Konvoj, 2. vydání, 2001. ISBN 80-7302-007-6
<ftp://math.feld.cvut.cz/pub/olsak/tbn/tbn.pdf>
- [7] Polák, J. *Přehled středoškolské matematiky*. Praha: Prometheus, 2000. ISBN 80-7196-196-5
- [8] Rybička, J. *L^AT_EX pro začátečníky*. Brno: Konvoj, 2. vydání, 1999. ISBN 80-85615-74-6

Zajímavé odkazy týkající se tématu:

- <http://cm.bell-labs.com/who/hobby/MetaPost.html>
- <http://comp.uark.edu/~luecking/tex/mfpic.html>
- <http://ftp.cstug.cz/pub/tex/CTAN/graphics/mfpic/>
(bohužel zatím je zde pouze starší verze)
- <http://ftp.cstug.cz/pub/tex/CTAN/systems/knuth/mf/mfbook.tex>

Rejstřík

- ahangle, 51
- ahlength, 51
- arclength, 48
- arctime, 48

- background, 55
- bbox, 54
- bboxmargin, 54
- beginfig, 44, 46
- begingroup, 56, 57
- beveled, 51
- boolean, 41, 42
- btex, 53, 54
- buildcycle, 55
- butt, 51

- color, 41, 42
- currentpicture, 42
- cutafter, 48
- cutbefore, 47

- dashed, 50
- dashpattern, 51
- def, 43, 56
- defaultfont, 52, 53
- defaultpen, 46
- defaultscale, 53
- defaultscaled, 53
- direction, 48
- directionpoint, 48
- directiontime, 48
- dotlabel, 52
- downto, 59
- draw, 42, 46
 - controls, 46
 - curl, 46
 - cycle, 46, 55
 - dir, 46
 - tension, 46
 - drawarrow, 51
 - drawarrow reverse, 51
 - drawdblarrow, 51
 - drawdot, 46

- else, 57
- elseif, 57
- end, 44
- enddef, 56, 60
- endfig, 44
- endfor, 58–60
- endgroup, 56, 57
- etex, 53, 54
- evenly, 50
- exitif, 60
- exitunless, 60
- expr, 58, 60

- fi, 57
- fill, 55
- filldraw, 55
- fontsize, 53
- for, 58–60
- forever, 60
- forsuffixes, 60
- fullcircle, 49

- halfcircle, 49

- if, 57
- image, 60
- infinity, 46
- intersectionpoint, 47
- intersectiontimes, 47

- label, 51, 52
- labeloffset, 52
- length, 47
- linecap, 51

linejoin, 51
 makepath, 46
 makepen, 46
 mitered, 51
 mpxbreak, 53
 numeric, 41, 42, 57
 pair, 41, 42
 path, 41, 42, 54
 pen, 41, 42, 54
 pencircle, 45
 pensquare, 46
 pickup, 46
 picture, 41, 42, 50, 52–54
 point, 47
 prologues, 44, 62
 quatercircle, 49
 reflectedabout, 48
 rotated, 48
 rotatedaround, 48
 rounded, 51
 save, 57
 scaled, 45, 48, 51
 shifted, 48, 51
 show, 42
 slanted, 48
 spark, 43
 squared, 51
 step, 59
 str, 58
 string, 41, 42
 subpath, 47
 suffix, 58, 60
 tag, 43
 text, 58
 thelabel, 52
 token, 43
 transform, 41, 42
 transformed, 48
 inverse, 49
 undraw, 46
 undrawdot, 46
 unfill, 55
 unfilldraw, 55
 until, 59
 upto, 59
 vardef, 56
 verbatimtex, 54
 whatever, 45
 withdots, 50
xpřívona, 42
xscaled, 48
xčíslo, 45
ypřívona, 42
yscaled, 48
yčíslo, 45
zpřívona, 42
zscaled, 48
zčíslo, 45

Summary: METAPOST and mfpic—the first part

This article is dedicated to introduction of the language used by METAPOST system by John D. Hobby. All commands of the language are shown in context and are illustrated by simple examples.