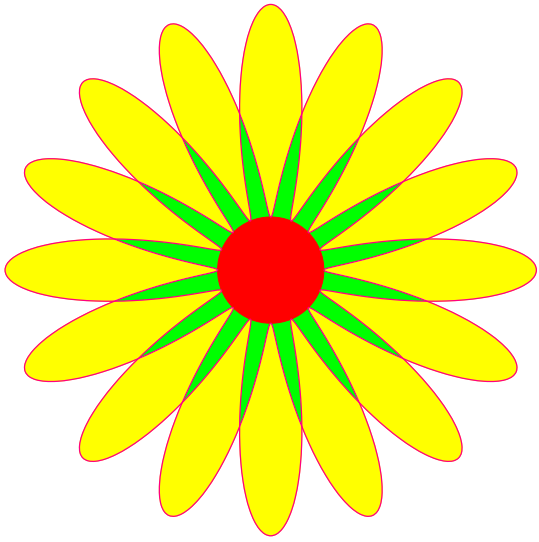
Masarykova Univerzita v Brně
Přírodovědecká Fakulta

Diplomová práce

Tvorba obrázků pro matematické texty pomocí METAPOSTU



Jméno a příjmení diplomantky: Miroslava Krátká

Brno 2001

Tvorba obrázků pro matematické texty pomocí METAPOSTU

Miroslava Krátká

Poděkování: Na tomto místě bych ráda poděkovala vedoucímu diplomové práce Doc. RNDr. Jaromíru Kubenovi, CSc. za jeho trpělivost, odborné rady a konzultace při tvorbě práce. Dále děkuji svým rodičům a sestře za vytvoření studijního prostředí, rodině Janíkových a zvláště panu Mgr. Pavlu Janíkovi za podporu a pomoc.

Prohlášení: Na přípravě diplomové práce jsem pracovala samostatně za použití uvedených zdrojů.

V Brně dne 17. dubna 2001

Obsah

Práce je věnována programovacímu jazyku METAPOST a balíku jeho maker `mfpic`. Text byl zpracován typografickým systémem LaTeX 2$_\varepsilon$ na operačním systému Linux.

Obrázky doplňující text jsou vloženy do obdélníků, respektive čtverců. Tyto nejsou součástí zdrojových kódů obrázků.

Práce je dostupná v elektronické podobně na adrese http://mirka.janik.cz/dp.

Přílohou diplomové práce je CD ROM, který obsahuje:

- tuto práci připravenou ke zpracování na operačním systému Linux
- PDF verzi práce
- `mfpic` verzi 0.4.05
- Zpravodaj Československého sdružení uživatelů TeXu 2/94

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 překladu 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 5.

1.1.1 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 3).

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

Ke zjištění typu proměnné nebo její hodnoty slouží příkaz

`show` *název_proměnné* .

1.1.2 Tokeny

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

Nejpoužívanějšími symbolickými tokeny jsou velká a malá písmena anglické abe-

cedy 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 17) nebo byl definován pomocí `def` jako makro (viz odstavec Makra na straně 17), se nazývá **spark**.

1.2 Vstupní soubor

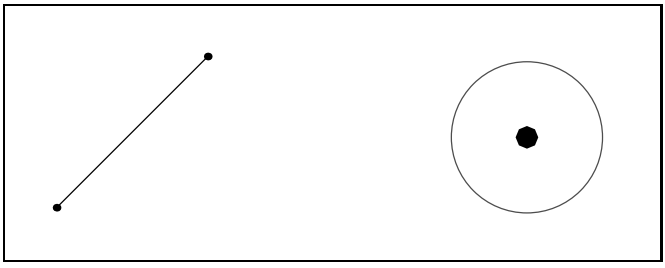
Vstupní soubor mívá obvykle příponu `.mp`. Soubor `obrazek.mp` může vypadat například takto:

```
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 na straně 24. 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 TEXovského dokumentu; ten zpracujeme odpovídajícím způsobem a prohlédneme vhodným prohlížečem. Vkládání obrázku je závislé na použitém formátu TEXu:

- v plain TEXu, $\mathcal{AMS}$-TEXu a LATEXu 2.09: \`epsfbox` {*jméno_souboru*} (potřebujeme soubor `epsf.tex`),
- v pdfTEXu: \`convertMPtoPDF` {*jméno_souboru*}{1}{1} (jsou nezbytné soubory `supp-pdf.tex` a `supp-mis.tex`),
- v LATEX 2$_\varepsilon$ a pdfLATEXu: \`includegraphics` {*jméno_souboru*} (předpokládá načtení balíků `graphics` nebo `graphicx`, pro pdfLATEX 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 pdfLATEXu 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

1.3.1 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). Následné `u=2cm` by dalo chybu, zatímco `u:=2cm` by změnilo hodnotu `u`.

1.3.2 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 `z`*číslo* reprezentuje bod o souřadnicích (`x`*číslo*,`y`*číslo*). 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.

1.3.3 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 opětnému 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.

1.3.4 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* .

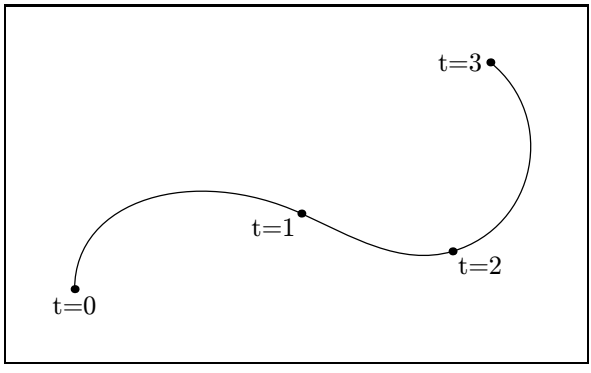
Číslo menší než 1 snižuje zakřivení, *číslo* větší než 1 jej zmenšuje. Poslední možností je přímé zadání kontrolních bodů příkazem

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

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

1.3.5 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` (t_1, t_2) `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 .

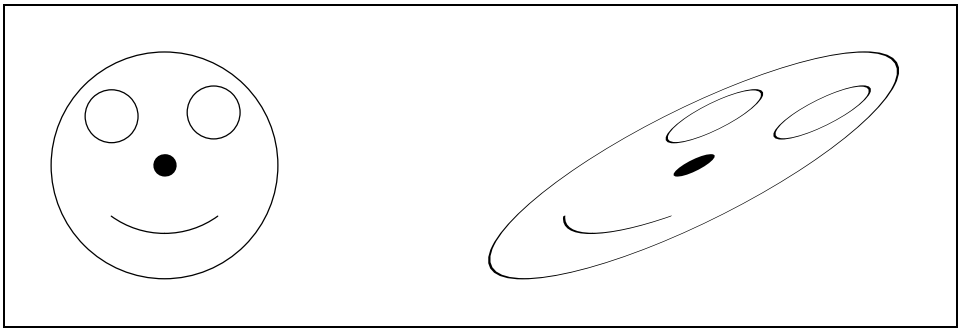
1.3.6 Afinní transformace

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` $\theta \longrightarrow (x \cos\theta - y \sin\theta, x \sin\theta + y \cos\theta)$,
- 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` $((a, b), \theta)$.

Použití transformací ukazuje obrázek 3.



Obrázek 3: Použití afinních transformací

```
u=cm;
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);
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:

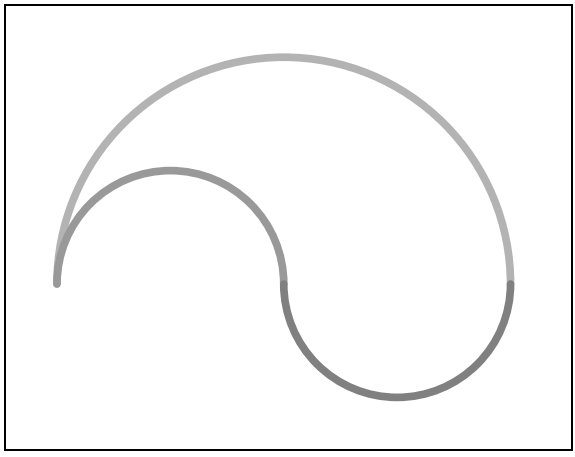
$p = q$ `transformed inverse` t .

1.3.7 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

1.3.8 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(`*seznam* `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`.

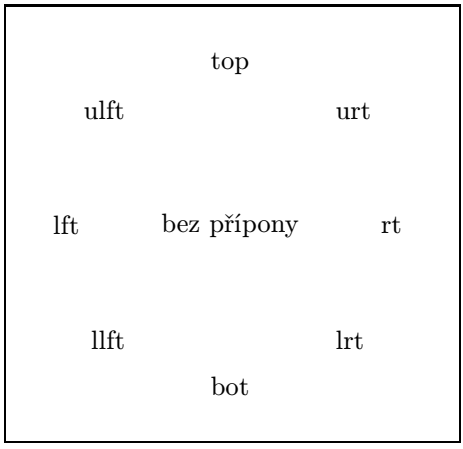
Podrobnosti a ukázky všech předešlých proměnných a příkazů v odstavci najdeme v manuálu k METAPOSTu ([2, str. 32]).

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*.



Obrázek 5: Přípony používané pro umisťování textu nebo obrázku

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

```
defaultfont:="csr10"; u:=cm; 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);
```

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 TEXovské 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;
```

1.4.1 Text v METAPOSTu

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

`btex` *příkazy_TEXu* `etex` ,

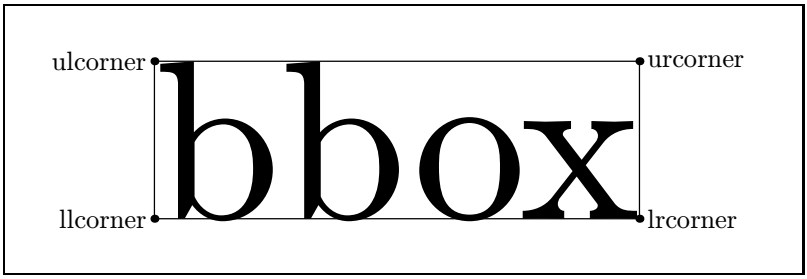
jsou *příkazy_TEXu* vysázeny TEXem a výsledek je předán zpět METAPOSTu jako datový typ `picture`, což nám umožňuje text otáčet a jinak přizpůsobovat našim požadavkům.

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 popsaný 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 TEX 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, tj. 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.

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



Obrázek 6: Bounding box

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.

1.4.2 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 přednastavenu 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

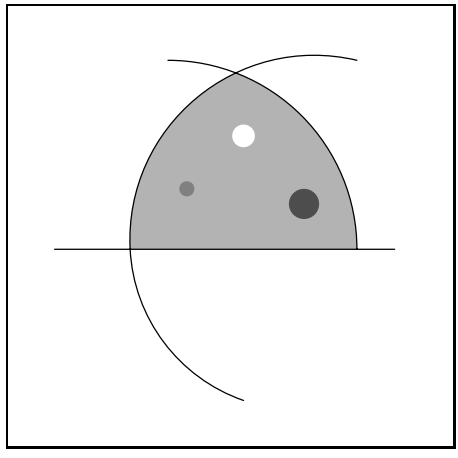
`filldraw` *cesta* `withcolor` *barva* ,

respektive příkaz `unfilldraw` *cesta* pro odbarvování.

Příkaz

`buildcycle` (p_1, p_2, ..., p_k)

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 nejdřívějším 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 nadefinováno 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 `begingroup` ... `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.

1.6.1 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 předcházejích 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.

1.6.2 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;
```

1.6.3 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 ukázku 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.

1.6.4 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* = *výraz*$_1$, *výraz*$_2$, ..., *výraz*$_n$:
> *posloupnost_příkazů*
> `endfor` .

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

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

> `for` *posloupnost_tokenů* = *výraz*$_1$ `step` *krok* `until` *výraz*$_2$:
> *posloupnost_příkazů*
> `endfor` .

Cyklus nejprve vyhodnotí, zda jsou *výrazu*$_1$, *výrazu*$_2$ a *kroku* přiřazeny nějaké konkrétní numerické hodnoty. Dále postupuje takto:
Je-li *krok* > 0 a *výraz*$_1$ $>$ *výraz*$_2$, nebo *krok* < 0 a *výraz*$_1$ $<$ *výraz*$_2$, *posloupnost_příkazů* není provedena. V případě, že ani jedna z předcházejících podmínek splněna není, je provedena *posloupnost_příkazů* pro hodnotu *výrazu*$_1$. *Výraz*$_1$ je nahrazen hodnotou (*výraz*$_1$+*krok*). Tento proces se opakuje s novou hodnotou *výrazu*$_1$.

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` .

1.6.5 Užitečné makro

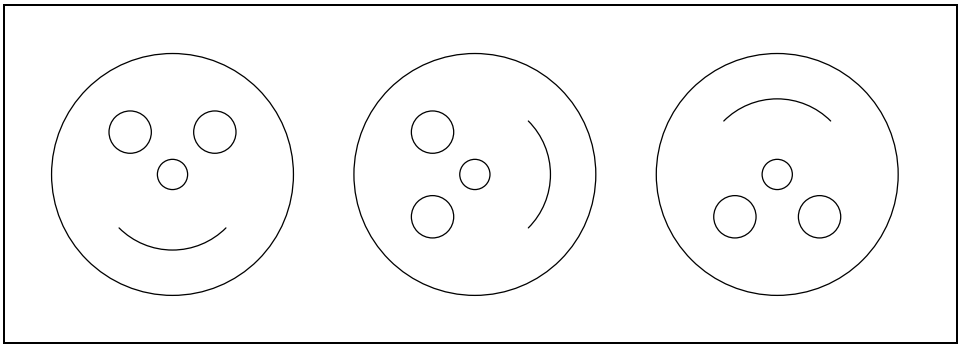
Pěkným a užitečným makrem je makro `image` , které najdeme v již zmíněném souboru `plain.mp`:

```
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*) ,

přičemž *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:=.8cm;

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 3 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 vyrastrování 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 TEXovského 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, $c \leqq 0$ to, že obrázek není oříznut (při c>0 je oříznut na nejmenší možný obdélník), což nám při vkládání do textu nevadí, 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.

2 mfpic

První část práce ukazuje nemalé možnosti METAPOSTu při kreslení obrázků. Chceme-li ovšem nakreslit náročnější obrázek, nestačí jen prolistovat manuálem; musíme se naučit pracovat s novým programovacím jazykem. To mnohé, zejména ty méně zvídavé, odradí. A právě pro ně se objevují balíky maker pro METAPOST.

Takovými balíky jsou `mfpic`, kterým se v dalším textu budeme podrobněji zabývat (http://comp.uark.edu/~luecking/tex/mfpic.html), a `feynmf`, jehož autorem je Thorsten Ohl
(ftp://ftp.cstug.cz/pub/CTAN/macros/latex/contrib/supported/feynmf).
V roce 1989 byl vytvořen soubor maker `feynman.mf` pro kreslení Feynmanových diagramů. O pět let později, tedy v roce 1994, se autor inspiroval balíkem `mfpic` (zdrojový kód se zapisoval do zdrojového souboru LaTeXu). Vzniklý `feynmf` využíval pro kreslení obrázků buď METAFONT, nebo METAPOST.

Zcela odlišná je grafická nadstavba pro METAPOST nazvaná `metagraf`. Jde o aplikaci napsanou v jazyce Java (je třeba Java2). Kreslí se pomocí menu a myši (http://w3.mecanica.upm.es/metapost/metagraf.php).

2.1 Historie `mfpic`

Během roku 1992 začal Tom Leathrum pracovat na tvorbě `mfpic`. Tento balík maker využívá pro kreslení METAFONT, přičemž zdrojový kód se zapisuje do vstupního souboru TeXu a užití příkazů samotného METAFONTu je pro uživatele skryto. V roce 1996 vývoj převzal Daniel H. Luecking. Od verze 0.3.0 z roku 1998 již můžeme využívat i METAPOST. Aktuální verzí k dubnu 2001 je verze 0.4.05. První zmínka o `mfpic` v češtině se objevila v roce 1994 v druhém čísle Zpravodaje CSTUGu zmíněného roku ([5, str. 87]). Autor se věnoval verzi `mfpic` 0.2, kterou upravil, a verzi 0.2.5.

2.2 Zdrojový soubor

Makra `mfpic` lze použít ve formátech plain, LaTeX i $\mathcal{A}\mathcal{M}\mathcal{S}$-TeX. Než začneme tvořit naše obrázky, potřebujeme mít nainstalované následující soubory a balíky (prvním předpokladem je samozřejmě TeX a METAPOST, respektive METAFONT):

- `grafbase.mp` a `dvipsnam.mp` v adresáři prohledávaném METAPOSTem (například aktuální adresář), respektive `grafbase.mf` v adresáři prohledávaném METAFONTem,
- `mfpic.tex` a `mfpic.sty` v adresáři, ve kterém hledá TeX (například aktuální adresář),

- `epsf.tex` pro plain TEX, $\mathcal{AMS}$-TEX a LATEX2.09,
- `supp-pdf.tex` a `supp-mis.tex` pro pdfTEX,
- `graphics` nebo `graphicx` pro LATEX 2_ε a pro pdfLATEX s výstupem do pdf; tento případ vyžaduje soubory `pdftex.def`, `supp-pdf.tex`, `supp-mis.tex`.

Jestliže žádný z uvedených souborů a balíků nenačteme (nebo tak učiníme až po načtení balíku `mfpic`), `mfpic` to provede sám. Například pro LATEX načítá balík `graphics`. Toto vše je nezbytné pro vkládání námi vytvořených obrázků, jež je prováděno takto:

- `\epsfbox` {*jméno_souboru*} v plain TEXu, $\mathcal{AMS}$-TEXu a LATEXu 2.09,
- `\convertMPtoPDF` {*jméno_souboru*}{1}{1} v pdfTEXu,
- `\includegraphics` {*jméno_souboru*} v LATEX 2_ε a pdfLATEXu.

V dalším výkladu budeme pojmem LATEX myslet LATEX 2_ε. Zaměříme se na `mfpic` s využitím METAPOSTu.

Jak víme z předchozích odstavců, METAPOST vytváří soubory s příponou *.číslo*, a proto je v pdfLATEXu dále třeba přidat příkaz

`\DeclareGraphicsRule` {∗} {`mps`} {∗} {} .

Každý soubor využívající `mfpic` s METAPOSTem má v LATEXu následující strukturu:

```
\documentclass{article}
\usepackage[metapost]{mfpic}%%% pokud neuvedeme nepovinný para-
                            %%% metr, bude použit pro tvorbu
                            %%% obrázků Metafont
%\usepackage{mfpic}  %%% tyto dva řádky jsou ekvivalentní s řád-
%\usemetapost        %%% kem předcházejícím

\begin{document}

\opengraphsfile{obrazek}

%libovolný zdrojový kód

\begin{mfpic}[a][b]{c}{d}{e}{f}
%příkazy popisující obrázek - výsledkem je obrazek.1
\end{mfpic}
```

```
%okolí mfpic lze zapisovat také takto:

\mfpic[g][h]{i}{j}{k}{l}
%příkazy popisující obrázek - výsledkem je obrazek.2
\endmfpic

\closegraphsfile

\opengraphsfile{picture}

\mfpic[m][n]{o}{p}{q}{r}
%příkazy popisující obrázek - výsledkem je picture.1
\endmfpic

\closegraphsfile

\end{document}
```

Obdobný zdrojový soubor v plain TEXu:

```
\input mfpic.tex
\usemetapost

\opengraphsfile{obrazek}

\mfpic[a][b]{c}{d}{e}{f}
%příkazy popisující obrázek - výsledkem je obrazek.1
\endmfpic

\closegraphsfile

\bye
```

Příkazem

\opengraphsfile{*název_souboru*}

otevíráme soubor, do kterého se zapisují příkazy METAPOSTu, příkaz

\closegraphsfile

tento soubor zavírá (tyto příkazy by měly být používány pouze v udaném tvaru, to jest ne jako okolí \begin{*okolí*} ... \end{*okolí*} v LATEXu). Příkazy

`\mfpic` [*měřítko*$_{\text{na\_ose\_x}}$] [*měřítko*$_{\text{na\_ose\_y}}$] {$x_{\min}$} {$x_{\max}$} {$y_{\min}$} {$y_{\max}$}
...
`\endmfpic`

`\begin{mfpic}`[*měřítko*$_{\text{na\_ose\_x}}$][*měřítko*$_{\text{na\_ose\_y}}$]{$x_{\min}$}{$x_{\max}$}{$y_{\min}$}{$y_{\max}$}
...
`\end{mfpic}` (pouze v LaTeXu)

uzavírají popis jednoho obrázku. Měřítko musí být zadáno alespoň na jedné ose; je-li zadáno pouze jedno, je měřítko shodné pro osu x i y. Čísla $x_{\min}$, $x_{\max}$ určují rozsah osy x, čísla $y_{\min}$, $y_{\max}$ určují rozsah osy y.

Nyní jsou dvě možnosti způsobu zpracování vstupního souboru:

Po překladu vstupního souboru LaTeXem (respektive plain TeXem) vzniknou soubory `obrazek.mp` a `picture.mp`. Tyto soubory přeložíme programem `mpost` (viz první část práce) a poté znovu spustíme LaTeX (respektive TeX). Výsledkem je soubor s příponou `.dvi`. Nejsou-li při jeho prohlížení viditelné obrázky (například `xdvi` na novějších verzích operačního systému Linux obrázky zobrazuje), vytvoříme příslušným programem (`dvips`) postscriptový soubor a prohlédneme si jej například pomocí Ghostview.

Použijeme-li pdfLaTeX (respektive pdfTeX), vznikne soubor s příponou `.pdf`, který si prohlédneme například prohlížečem Acrobat Reader.

2.3 Volby `mfpic`

Již v předchozím odstavci jsme se seznámili s jednou volbou `mfpic`, a to `metapost`. Jak bylo vidět z ukázky, tyto volby se zapisují buďto jako nepovinné parametry u LaTeXovského příkazu `\usepackage`, nebo ekvivalentními příkazy.

Můžeme použít následující volby:

- `metapost`, `\usemetapost` — popisy obrázků vytvářených pomocí `mfpic` jsou zapisovány do souboru s příponou `.mp` a tento soubor je zpracován METAPOSTem; příkaz musí být zapsán před otevřením souboru `.mp`, to jest před příkazem `\opengraphsfile`,

- `metafont`, `\usemetafont` — popisy obrázků jsou zadávány do souboru s příponou `.mf`, tento soubor je zpracován METAFONTem; příkaz musí být zapsán před otevřením souboru `.mf`, to jest před příkazem `\opengraphsfile`; tato volba je přednastavena (chceme-li tedy používat METAFONT, není nutno volbu udávat),

- `mplabels`, `\usemplabels`, `\nomplabels` — tato volba ovlivňuje zpracování popisek u příkazu `\tlabel`, popisky jsou zpracovány METAPOSTem; příkaz

musí být uveden až za příkazem \usemetapost ; používáme pouze s následující volbou,

- truebbox, \usetruebbox, \notruebbox — zajistí, aby METAPOST určil bounding box obrázku včetně textu; tuto volbu využíváme zároveň s volbou mplabels,

- clip, \clipmfpic, \noclipmfpic — odstraní části obrázku přesahující obdélník udaný rozměry u \mfpic; lze použít jak u METAFONTu, tak u METAPOSTu,

- centeredcaptions, \usecenteredcaptions, \nocenteredcaptions — zkrácené řádky textu pod obrázkem vytvořeného příkazem \tcaption jsou umístěny do středu, neboli vycentrovány podle šířky obrázku,

- debug, \mfpicdebugtrue, \mfpicdebugfalse — sdělí nám více informací o průběhu zpracování vstupního souboru; tyto informace jsou zapisovány do souboru s příponou .log a v některých případech na obrazovku.

Pro naši práci jsou ideální následující dvě kombinace:

```
\usepackage [metapost, mplabels, truebbox] {mfpic}   ,
\usepackage [metapost, mplabels, truebbox, clip] {mfpic}   .
```

Volba mplabels zajistí, aby popisky zpracovával METAPOST; volba truebbox zajistí, aby METAPOST určil přesný bounding box obrázku a tento předal TEXu (a tedy nevzniknou problémy například při obtékání obrázku); volba clip způsobí ořezání výsledného obrázku na rozměr námi zadaný (tudíž zamezíme zasahování okolního textu do obrázku). Všechny další kombinace různých voleb mfpic mohou dát zcela nesprávné výsledky.

2.4 Parametry mfpic

V makrech mfpic se samozřejmě objevuje mnoho parametrů, jejichž změnami můžeme dosáhnout různých, pro nás příjemných, efektů. U většiny maker, o kterých se zmiňuji v dalších odstavcích, je řečeno, jaké parametry ovlivňují jejich výsledky.

Některé parametry jsou uloženy jako dimenze TEXu; jiné jsou uloženy METAPOSTem, tedy jejich případné změny uvnitř okolí \mfpic ... \endmfpic mají pouze lokální význam.

Významné pro nás mohou být následující parametry (veškerá zde zmíněná makra jsou podrobněji popisována v dalším textu).

Parametry TEXu:

- `\mfpicunit` — představuje základní jednotku délky; přednastavena je na hodnotu 1 pt; všechny x-ové a y-ové souřadnice jsou jí v makrech `mfpic` násobeny (proměnná typu dimenze),

- `\pointsize` — uchovává průměr kružnice používané v makru `\point`, dále průměr symbolů v makrech `\plotsymbol` a `\plot`; přednastavena je hodnota 2 pt (proměnná typu dimenze),

- `\pointfilltrue`, `\pointfillfalse` — logická proměnná určující, zda bude kružnice kreslená příkazem `\point` vyplněna, či nikoli,

- `\headlen` — dimenze udávající délku šipky při použití makra `\arrow`; přednastavena je délka 3 pt,

- `\axisheadlen` — dimenze, která uchovává délku šipky při využití makra `\axes`, `\xaxis`, `\yaxis`; přednastavena je délka 5 pt,

- `\dashlen` — dimenze určující délku čárek v makru `\dashed`; přednastavena je délka 4 pt,

- `\dashspace` — dimenze udávající velikost mezer používaných v makru `\dashed`; přednastavena je mezera 4 pt,

- `\dashlineset` — makro, které nastaví standardní hodnoty pro `\dashlen` a `\dashspace`, to znamená obě hodnoty na 4 pt,

- `\dotlineset` — makro nastavující hodnotu pro `\dashlen` na 1 pt a hodnotu `\dashspace` na 2 pt,

- `\symbolspace` — dimenze udávající velikost mezery mezi symboly v makru `\plot`; přednastavena je mezera 5 pt,

- `\hashlen` — dimenze uchovávající délku čárek používaných v makrech `\xmarks` a `\ymarks`; přednastaveny jsou délky 4 pt,

- `\dotsize` — dimenze určující velikost kruhů v makru, `\dotted`; přednastaven je průměr 0,5 pt,

- `\dotspace` — dimenze udávající mezery mezi středy kruhů vykreslujících se v makru `\dotted`; přednastavena je mezera o velikosti 3 pt,

- `\polkadotspace` — dimenze nastavující velikost mezer mezi středy kruhů používaných v makru `\polkadot`; přednastavena je mezera 10 pt,

- `\hatchspace` — dimenze uchovávající vzdálenost rovnoběžek používaných v makru `\hatch`; přednastavena je vzdálenost 3 pt,

- `\mfpicheight` — dimenze, v níž je uložena výška obrázku vytvořeného v okolí `\mfpic` ... `\endmfpic`,
- `\mfpicwidth` — dimenze, v níž je uložena šířka obrázku vytvořeného okolím `\mfpic` ... `\endmfpic`.

Poslední dvě dimenze (výška a šířka obrázku) jsou pro nás praktické v případě, že bychom chtěli „šidit“ při umisťování obrázku. Standardně jsou vypočítávány z údajů zadaných v příkazu `\mfpic` v úvodu každého obrázku.

Chceme-li některou z dimenzí TEXu změnit, provedeme to obvyklým způsobem, to znamená například `\mfpicunit=3pt` (v LATEXu můžeme také příkazem `\setlength{\mfpicunit}{3pt}`).

Parametry METAPOSTu (neboli TEXovská makra nastavující interní parametry METAPOSTu, jejichž přesné názvy pro nás nejsou podstatné):

- `\pen` {*tloušťka*}, `\drawpen` {*tloušťka*} — nastaví tloušťku pera pro kreslení; přednastavena je hodnota 0,5 pt,
- `\hatchwd` {*tloušťka*} — určuje tloušťku čar používaných při šrafování; přednastavena je hodnota 0,5 pt,
- `\polkadotwd` {*průměr*} — přiřazuje průměr kruhům využívaným v makru `\polkadot`; přednastavena je hodnota 5 pt,
- `\headshape` {*poměr*} {*napětí*} {*vybarvení*} — toto makro určuje tvar šipky používané v makrech `\arrow`, `\axes`, `\xaxis`, `\yaxis`; *poměr* je poměr šířky šipky k její délce, *napětí* představuje napětí Bézierovy křivky, kterou je šipka vykreslena, a *vybarvení* značí, či je šipka otevřená (hodnota `false`) nebo uzavřená a vybarvená; přednastaveny jsou hodnoty 1, 1, `false`.

2.5 Barvy `mfpic`

V METAPOSTových makrech pro `mfpic`, to znamená v souboru `grafbase.mp`, se setkáváme se čtyřmi barvami — `drawcolor`, `fillcolor`, `hatchcolor`, `headcolor`.

Barvou `drawcolor` jsou vykreslovány křivky, barva `fillcolor` je využívána pro výplně, `hatchcolor` pro šrafování a `headcolor` pro kreslení šipek. Všechny tyto barvy jsou přednastaveny na černou barvu.

Nejjednodušší způsob, jak některou z výše uvedených barev změnit, je použití odpovídajícího příkazu:

`\drawcolor` {*barva*} ,
`\fillcolor` {*barva*} ,

\hatchcolor {*barva*} ,
\headcolor {*barva*} .

Parametrem *barva* může být:

- předdefinovaná barva — `black`, `white`, `red`, `green`, `blue`, `cyan`, `magenta`, `yellow` (tzn. černá, bílá, červená, zelená, modrá, tyrkysová, fialová, žlutá),
- barva vyjádřená modelem `rgb` — `rgb` (a, b, c), kde a, b, c jsou čísla z intervalu $\langle 0, 1\rangle$,
- barva vyjádřená modelem `cmyk` — `cmyk` (a, b, c, d), kde a, b, c, d jsou čísla z intervalu $\langle 0, 1\rangle$, převádí barvu udanou modelem `cmyk` na model `rgb`,
- barva vyjádřená modelem `RGB` — `RGB` (a, b, c), kde a, b, c jsou z intervalu $\langle 0, 255\rangle$, převádí barvu v modelu `RGB` na model `rgb`,
- barva vyjádřená modelem `gray` — `gray` (a), kde a je z intervalu $\langle 0, 1\rangle$, odpovídá barvě $a *$ `white`,
- barva námi předdefinovaná (model `named`) — můžeme využít barvy definované v souboru `dvipsnam.mp` nebo barvy jiné, námi nadefinované pomocí příkazu
  \mfpdefinecolor {*jméno_barvy*} {model} {barva} ,
  kde *jméno_barvy* je libovolný námi zvolený název pro barvu, *model* je jeden z modelů `rgb`, `RGB`, `cmyk`, `gray`, `named` a *barva* je vyjádření požadované barvy v použitém modelu. Použijeme-li model `named`, pouze přejmenujeme danou *barvu*.

Druhou možností změny barvy jsou příkazy:

\drawcolor [*model*] {*barva*} ,
\fillcolor [*model*] {*barva*} ,
\hatchcolor [*model*] {*barva*} ,
\headcolor [*model*] {*barva*} ,

kde *model* může být `rgb`, `cmyk`, `RGB`, `named`, nebo `gray` a *barva* je „hodnota“ barvy v příslušném modelu.

2.6 Jednoduché útvary

Příkaz

\pointdef {*název*} (x, y)

umožňuje definovat název pro bod a jeho souřadnice, kde *název* je námi zvolený název (neobsahující zpětné lomítko) pro bod o souřadnicích (x, y). Nadefinujeme-li například `\pointdef{M}(3,7)`, `mfpic` bude příkaz `\M` chápat jako zadání souřadnic (3,7) a `\Mx`, popř. `\My`, bude expandovat na 3, popř. 7.

Makro

`\point` [*průměr*] $\{(x_1, y_1), (x_2, y_2), \ldots\}$

vykreslí kruhy se středy $(x_1, y_1), (x_2, y_2), \ldots$ a průměrem odpovídajícím hodnotě proměnné `\pointsize` (přednastavené na velikost 2 pt). Nepovinný parametr *průměr* nám umožňuje přednastavenou hodnotu průměru kruhů jednoduše upravovat. Chceme-li body označovat pouze kružnicemi, využijeme příkazu `\pointfillfalse`; standardní nastavení, `\pointfilltrue`, vyplňuje kruhy barvou `fillcolor`.

Příkaz

`\grid`{*mezera_vertikálních_rovnoběžek*, *mezera_horizontálních_rovnoběžek*}

vytvoří systém kruhů, které umístí na průsečíky horizontálních a vertikálních rovnoběžek. Vzdálenost těchto rovnoběžek udáme dvěma povinnými parametry, a to *mezera_vertikálních_rovnoběžek* a *mezera_horizontálních_rovnoběžek*; tyto parametry zadáváme bezrozměrné, tedy pouze čísla bez jednotky. Kruhy odpovídají interní proměnné `onedot` typu `picture`, což je vnitřek kruhu o průměru 0,5 pt.

Nejsme-li zastánci označování bodů kruhy či kružnicemi, zalíbí se nám makro

`\plotsymbol` [*průměr*] {*symbol*} $\{(x_1, y_1), (x_2, y_2), \ldots\}$

vykreslující *symboly* se středy o souřadnicích $(x_1, y_1), (x_2, y_2), \ldots$ K dispozici jsou tyto *symboly*:

- `Circle` — kružnice,
- `Diamond` — kosočtverec,
- `Square` — čtverec,
- `Triangle` — trojúhelník,
- `SolidCircle` — kruh,
- `SolidDiamond` — vyplněný kosočtverec,
- `SolidSquare` — vyplněný čtverec,
- `SolidTriangle` — vyplněný trojúhelník,
- `Cross` — křížek,

- `Plus` — plus,
- `Star` — hvězda.

Velikost symbolů koresponduje s hodnotou `\pointsize` (=2 pt) a lze ji upravit nepovinným parametrem *průměr*. Vyplněné symboly mají barvu odpovídající `fillcolor`.

Lomenou čáru s vrcholy v bodech $(x_1, y_1), (x_2, y_2), \ldots$ nakreslíme příkazy

`\polyline` $\{(x_1, y_1), (x_2, y_2), \ldots\}$,
`\lines` $\{(x_1, y_1), (x_2, y_2), \ldots\}$.

Makro

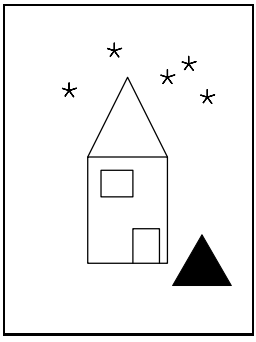
`\polygon` $\{(x_1, y_1), (x_2, y_2), \ldots\}$

vykreslí uzavřený polygon s vrcholy v bodech $(x_1, y_1), (x_2, y_2), \ldots$

Příkaz

`\rect` $\{(x_1, y_1), (x_2, y_2)\}$

vytvoří obdélník, kde $(x_1, y_1), (x_2, y_2)$ jsou protilehlé vrcholy obdélníku.



```
\mfpic[10]{0}{7}{0}{10}
\rect{(2,1.5),(5,5.5)}
\polyline{(2,5.5),(3.5,8.5),(5,5.5)}
\lines{(3.7,1.5),(3.7,2.8),
                (4.7,2.8),(4.7,1.5)}
\polygon{(2.5,4),(2.5,5),
                    (3.7,5),(3.7,4)}
\plotsymbol[17pt]{SolidTriangle}
                           {(6.3,1.3)}
\plotsymbol[5pt]{Star}{(1.3,8),
 (3,9.5),(5,8.5),(6.5,7.75),(5.8,9)}
\endmfpic
```

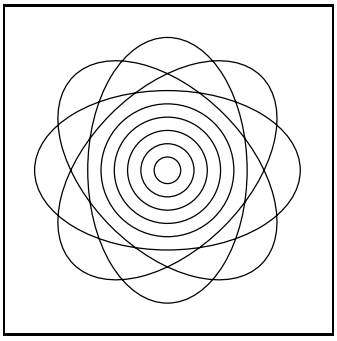
Kružnici se středem (x, y) a poloměrem r nakreslíme pomocí příkazu

`\circle` $\{(x, y), r\}$.

Elipsu popisuje makro

`\ellipse` $[\theta]$ $\{(x, y), r_\mathrm{x}, r_\mathrm{y}\}$,

kde r_x je délka hlavní poloosy, r_y délka vedlejší poloosy, (x, y) je střed elipsy a nepovinný parametr θ umožňuje otočení elipsy o θ stupňů proti směru hodinových ručiček kolem jejího středu.



```
\mfpic[10]{-5}{5}{-5}{5}
\circle{(0,0),2.5}
\circle{(0,0),2}
\circle{(0,0),1.5}
\circle{(0,0),1}
\circle{(0,0),0.5}
\ellipse{(0,0),5,3}
\ellipse[45]{(0,0),5,3}
\ellipse[90]{(0,0),5,3}
\ellipse[135]{(0,0),5,3}
\endmfpic
```

Bézierovu křivku procházející body $(x_1, y_1), (x_2, y_2), \ldots$ vykreslíme příkazem

\curve [*napětí*] $\{(x_1, y_1), (x_2, y_2), \ldots\}$.

Nepovinný argument *napětí* upravuje napětí křivky, o němž jsme se zmiňovali již v první části této práce (viz strana 6). Jeho předdefinovaná hodnota je 1.

Uzavřenou Bézierovu křivku procházející body $(x_1, y_1), (x_2, y_2), \ldots$ získáme příkazem

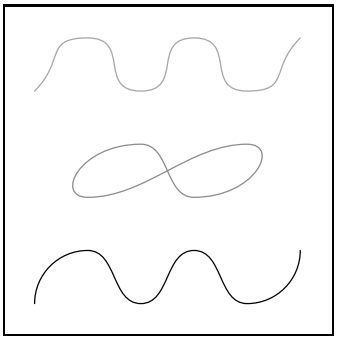
\cyclic [*napětí*] $\{(x_1, y_1), (x_2, y_2), \ldots\}$.

Nepovinný argument *napětí* využijeme stejně jako u předchozího příkazu v případě, že chceme upravit vzhled křivky.

Makro

\fcncurve [*napětí*] $\{(x_1, y_1), (x_2, y_2), (x_3, y_3)\}$

vytvoří křivku procházející danými body. Výsledná křivka je více symetrická než při použití \curve, příkaz je proto vhodný například pro kreslení grafů funkcí. Nepovinný argument opět ovlivňuje napětí křivky a je přednastaven na hodnotu 1,2.



```
\mfpic[20]{0}{5}{0}{5}
\curve{(0,0),(1,1),(2,0),
       (3,1),(4,0),(5,1)}
\draw[.55white]\cyclic{
       (1,2),(2,3),(3,2),
                   (4,3)}
\draw[.65white]\fcncurve{
       (0,4),(1,5),(2,4),
       (3,5),(4,4),(5,5)}
\endmfpic
```

Chceme-li nakreslit kruhový oblouk, máme k dispozici několik možností pro jeho zadání:

`\arc [s]` $\{(x_1,y_1),(x_2,y_2),\theta\}$

vykresluje oblouk z bodu (x_1,y_1) do bodu (x_2,y_2) tak, aby oblouk pokrýval úhel θ, který je měřen proti směru hodinových ručiček.

Oblouk procházející třemi body $(x_1,y_1),(x_2,y_2),(x_3,y_3)$ je zadán příkazem

`\arc [t]` $\{(x_1,y_1),(x_2,y_2),(x_3,y_3)\}$.

Makro

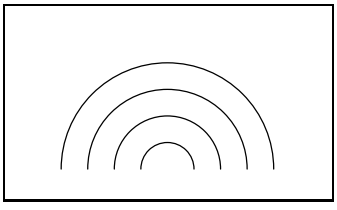
`\arc [p]` $\{(x_s,y_s),\theta_1,\theta_2,r\}$

nakreslí část kružnice se středem (x_s,y_s) a poloměrem r začínající úhlem θ_1 a končící úhlem θ_2, přičemž oba úhly jsou měřeny proti směru hodinových ručiček od kladné části souřadné osy x.

Posledním možným zadáním kruhového oblouku je příkaz

`\arc [c]` $\{(x_s,y_s),(x_1,y_1),\theta\}$,

který vykreslí oblouk se středem (x_s,y_s) začínající v bodu (x_1,y_1) a pokrývající úhel θ. Všechna předchozí zadání oblouku (s parametry `[s]`, `[t]`, `[p]`) jsou interně převáděna a vykreslena tímto makrem.



```
\mfpic[10]{-5}{5}{0}{5}
\arc[s]{(1,0),(-1,0),180}
\arc[t]{(-2,0),(0,2),(2,0)}
\arc[p]{(0,0),0,180,3}
\arc[c]{(0,0),(4,0),180}
\endmfpic
```

2.6.1 Souřadné osy

Pro znázornění souřadných os můžeme použít jedno z následujících maker:

`\axes` [*délka_šipky*] ,
`\xaxis` [*délka_šipky*] ,
`\yaxis` [*délka_šipky*] .

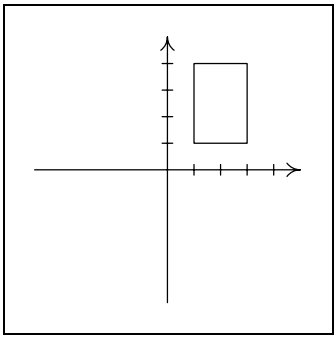
První z maker vyznačuje obě souřadné osy, zbývající dvě makra pouze souřadnou osu x, respektive y. Nevyužijeme-li nepovinný parametr, budou na osách vykresleny šipky o délce přiřazené proměnné `\axisheadlen` (přednastavena je hodnota 5 pt), tvaru určeného makrem `\arrow` a barvě odpovídající `headcolor`. Značky na osách získáme příkazem

`\xmarks` {*seznam_bodů_na_ose_x*}

pro osu x a příkazem

\ymarks {*seznam_bodů_na_ose_y*}

pro osu y. Délka čárečky odpovídá hodnotě \hashlen, nastavena je na 4 pt.



```
\mfpic[10]{-5}{5}{-5}{5}
\axes
\xmarks{1,2,3,4}
\ymarks{1,2,3,4}
\rect{(1,1),(3,4)}
\endmfpic
```

2.7 Polární souřadnice

Příkaz

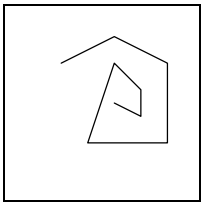
\plr {$(r_1,\theta_1),(r_2,\theta_2),\ldots$}

nahradí seznam bodů zadaných polárními souřadnicemi odpovídajícími souřadnicemi pravoúhlými.

Makro

\turtle {$(x,y),(u_1,v_1),(u_2,v_2),\ldots$}

nakreslí lomenou čáru, která vychází z bodu (x,y) a dále postupuje vždy o daný vektor.

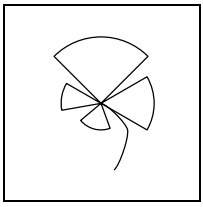


```
\mfpic[10]{0}{5}{0}{5}
\turtle{(1,4),(2,1),(2,-1),(0,-3),
   (-3,0),(1,3),(1,-1),(0,-1),(-1,0.5)}
\endmfpic
```

Kruhovou výseč o středu (x,y) a poloměru r zadáváme dvěma ohraničujícími úhly, a to pomocí příkazu

\sector {$(x,y),r,\theta_1,\theta_2$} .

Úhly θ_1 a θ_2 jsou měřeny proti směru hodinových ručiček od rovnoběžky se souřadnou osou x.



```
\mfpic[10]{0}{5}{0}{5}
\sector{(2.5,2.5),2,30,-30}
\sector{(2.5,2.5),2.5,45,135}
\sector{(2.5,2.5),1.5,150,190}
\sector{(2.5,2.5),1,220,290}
\curve[2]{(2.5,2.5),(3.5,1.5),(3,0)}
\endmfpic
```

2.8 Uložení objektu

Používáme-li ve zdrojovém textu obrázku několikrát jeden *objekt*, který je dvojicí nebo cestou, je výhodné si jej uložit příkazem

\store {*jméno*} {*objekt*}

a na potřebném místě vložit pomocí makra

\mfobj {*jméno*} .

Pomocí \mfobj lze vložit libovolnou proměnnou typu path (nejen uloženou pomocí \store), která je využívána v METAPOSTovém kódu našeho obrázku.

Praktické využití těchto příkazů bude ukázáno u vhodných obrázků následujících odstavců.

2.9 Prefixová makra

Některá z maker balíku mfpic se chovají jako tzv. *prefixová* makra, což v praxi znamená, že se tato makra aplikují na příkazy stojící ihned za nimi.

2.9.1 Kreslení

Makro

\draw [*barva*] ...

vykreslí námi požadovaný objekt plnou čarou.

Příkaz

\dashed [*délka*, *mezera*] ...

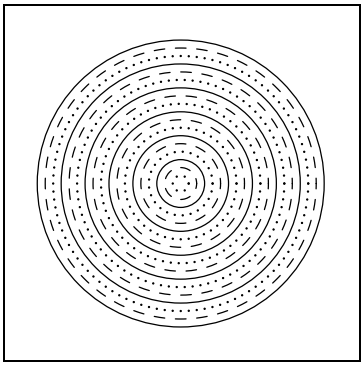
použije pro vykreslení objektu čáru přerušovanou. Přednastavená délka jedné čárky je dána hodnotou proměnné \dashlen (je rovna 4 pt); mezery mezi jednotlivými čárkami jsou určeny rozměrem \dashspace (je roven rovněž 4 pt). Čárky na začátku a konci křivky jsou dlouhé polovinu z hodnoty \dashlen. Aby byl zachován námi požadovaný poměr čárek a mezer, mohou být námi zadané délky

čárek a rozměry mezer v nepovinných parametrech mírně upraveny, tedy mohou být prodlouženy či zkráceny.

Křivku vykreslíme tečkovanou čarou, použijeme-li makro

\dotted [*velikost*, *mezera*] ... .

Předdefinovanou velikost teček udává hodnota proměnné \dotsize (je rovna 0,5 pt), předdefinovaný rozměr mezer je dán hodnotou \dotspace (je roven 3 pt). Na začátku a konci křivky je vykreslena tečka. Velikost teček a mezer může být ovlivněna stejně jako u \dashed.



```
\mfpic[10]{-5.5}{5.5}{-5.5}{5.5}
\dotted\circle{(0,0),0.3}
\dashed\circle{(0,0),0.6}
\draw\circle{(0,0),0.9}
\dotted\circle{(0,0),1.2}
\dashed\circle{(0,0),1.5}
\draw\circle{(0,0),1.8}
\dotted\circle{(0,0),2.1}
\dashed\circle{(0,0),2.4}
\draw\circle{(0,0),2.7}
\dotted\circle{(0,0),3}
\dashed\circle{(0,0),3.3}
\draw\circle{(0,0),3.6}
\dotted\circle{(0,0),3.9}
\dashed\circle{(0,0),4.2}
\draw\circle{(0,0),4.5}
\dotted\circle{(0,0),4.8}
\dashed\circle{(0,0),5.1}
\draw\circle{(0,0),5.4}
\endmfpic
```

Zdá-li se nám tečkovaná nebo čárkovaná čára příliš obyčejná, máme možnost vykreslení křivky například pomocí symbolů trojúhelníku, kosočtverce, křížku či hvězdičky

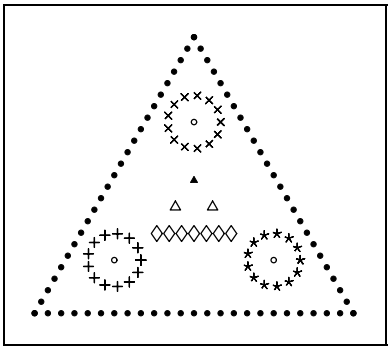
\plot [*velikost*, *mezera*] {*symbol*} ... .

Na místě povinného parametru *symbol* smí být jakýkoli symbol používaný u makra \plotsymbol (viz strana 34). Přednastavená *velikost* odpovídá \pointsize (=2 pt) a *mezera* odpovídá \symbolspace (=5 pt).

Makro

\plotnodes [*velikost*] {*symbol*} ...

také vykresluje *symboly* stejné jako v makru \plotsymbol (viz strana 34), ale nakreslí je pouze v bodech, jimiž jsme zadali křivku, na kterou toto makro aplikujeme. Parametr *velikost* je nastaven na hodnotu rovnu \pointsize (=2 pt).



```
\mfpic[10]{0}{12}{0}{10.4}
\plot{SolidCircle}
       \lines{(0,0),(12,0)}
\plot{SolidCircle}
  \lines{(0,0),(6,sqrt108)}
\plot{SolidCircle}
 \lines{(12,0),(6,sqrt108)}
\plot[3pt,5pt]{Plus}
           \circle{(3,2),1}
\plot[3pt,5pt]{Star}
           \circle{(9,2),1}
\plot[3pt,5pt]{Cross}
  \circle{(6,(2+sqrt27)),1}
\plotnodes{Circle}\polygon{
(3,2),(9,2),(6,(2+sqrt27))}
\plot[4pt,5pt]{Diamond}
    \lines{(4.6,3),(7.4,3)}
\plotnodes[3pt]{Triangle}
    \lines{(5.3,4),(6.7,4)}
\plotsymbol[2pt]
     {SolidTriangle}{(6,5)}
\endmfpic
```

Pro nadefinování vlastní čárkované čáry použijeme příkaz

\dashpattern {*jméno*} {*čárka*$_1$, *mezera*$_1$, *čárka*$_2$, *mezera*$_2$, ...} ,

kde *jménu* je přiřazen vzorek tvořený postupně *čárkou*$_1$, *mezerou*$_1$, *čárkou*$_2$, *mezerou*$_2$ atd. *Čárka* délky 0 pt odpovídá tečce. Popis vzoru musí končit *mezerou*.

Námi nadefinovanou čáru využijeme pro vykreslení křivky pomocí příkazu

\gendashed {*jméno*} ... .

```
\mfpic[10]{0}{5}{0}{5}
\dashpattern{moje}{0pt,1pt,2pt,3pt,4pt,
                   5pt,6pt,7pt,8pt,9pt}
\gendashed{moje}\lines{(0,0),(5,0)}
\pen{2pt}
\dashpattern{takemoje}{0pt,3pt,4pt,7pt}
\gendashed{takemoje}\lines{(0,2),(5,2)}
\pen{4pt}
\dashpattern{jestemoje}{0pt,5pt,7pt,
                                   10pt}
\gendashed{jestemoje}\lines{(0,4.5),
                                 (5,4.5)}
\endmfpic
```

2.9.2 Výplně

Následující *prefixová* makra mohou být aplikována pouze na uzavřené křivky.

Makro

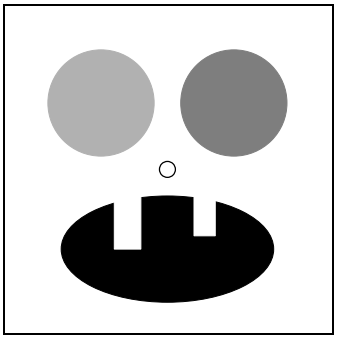
\gfill [*barva*] ...

vyplní uzavřenou křivku barvou uvedenou v nepovinném parametru *barva*. Není-li *barva* uvedena, je použita přednastavená barva odpovídající `fillcolor`. Inverzním příkazem ke \gfill je příkaz \gclear.

Příkaz

\shade [*šeď*] ...

vyplní uzavřenou křivku. Nepovinným parametrem *šeď* můžeme ovlivňovat stupeň šedi (přednastavena je hodnota 0.75`white`).



```
\mfpic[10]{-5}{5}{-5}{5}
\gfill\ellipse{(0,-3),4,2}
\gclear\rect{(-2,-3),(-1,-1)}
\gfill[white]
     \rect{(1,-2.5),(1.8,-1)}
\shade[0.9]\circle{(-2.5,2.5),2}
\shade[0.7]\circle{(2.5,2.5),2}
\circle{(0,0),0.3}
\endmfpic
```

Makro

\thatch [*mezera*,*úhel*] [*barva*] ...

vyšrafuje uzavřenou křivku. Mezery, standardně nastavené na hodnotu \hatchspace (=3 pt), mohou být změněny hodnotou nepovinného parametru *mezera*. Při nulové či záporné hodnotě je křivka vyplněna stejně jako příkazem \gfill. Tloušťka čáry, kterou jsou šrafy vykreslovány, je dána makrem \hatchwd (standardně =0,5 pt). Parametrem *úhel* ovlivníme sklon šrafů (přednastaven je úhel 0°) a parametrem *barva* jejich barvu (ta odpovídá hodnotě hatchcolor). Parametry *mezera* a *úhel* zadáváme pouze současně a také při využití parametru *barva* musí být zadány i *mezera* a *úhel*.

Pro práci se šrafováním jsou připravena makra:

\lhatch [*mezera*] [*barva*] ...

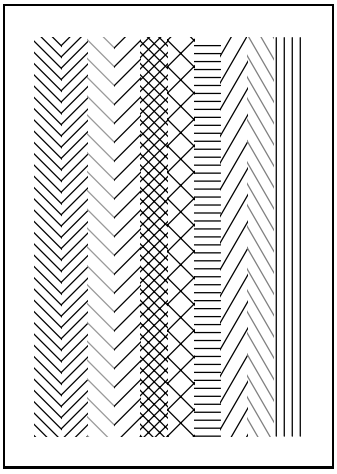
šrafuje z levého horního rohu k pravému dolnímu (to znamená pod úhlem −45°);

\rhatch [*mezera*] [*barva*] ...

šrafuje z pravého horního k levému dolnímu rohu (to znamená pod úhlem 45°);

\xhatch [*mezera*] [*barva*] ...
\hatch [*mezera*] [*barva*] ...

jsou kombinací lhatch a rhatch.

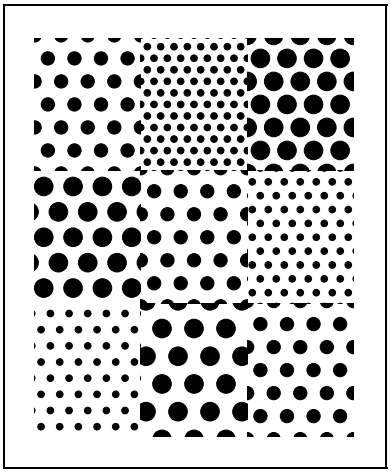


```
\mfpic[10]{0}{10}{0}{15}
\lhatch\rect{(0,0),(1,15)}
\rhatch\rect{(1,0),(2,15)}
\lhatch[5][.6white]
       \rect{(2,0),(3,15)}
\rhatch[5]\rect{(3,0),(4,15)}
\xhatch\rect{(4,0),(5,15)}
\hatch[7]\rect{(5,0),(6,15)}
\thatch\rect{(6,0),(7,15)}
\thatch[5,60 deg]
       \rect{(7,0),(8,15)}
\thatch[4,120 deg][.5white]
       \rect{(8,0),(9,15)}
\thatch[3,90 deg]
      \rect{(9,0),(10,15)}
\endmfpic
```

Makro

\polkadot [*mezera*] ...

vyplní uzavřený útvar kruhy o velikosti udané hodnotou \polkadotwd (=5 pt) vyplněnými barvou fillcolor s mezerami odpovídající hodnotě \polkadotspace (=10 pt).



```
\mfpic[10]{0}{12}{0}{15}
\polkadot\rect{(0,10),(4,15)}
\polkadot\rect{(4,5),(8,10)}
\polkadot\rect{(8,0),(12,5)}
\polkadotwd{2.5pt}
\polkadot[5]
   \rect{(4,10),(8,15)}
\polkadot[6]
   \rect{(8,5),(12,10)}
\polkadot[7]\rect{(0,0),(4,5)}
\polkadotwd{7pt}
\polkadot\rect{(8,10),(12,15)}
\polkadot[11]
   \rect{(0,5),(4,10)}
\polkadot[12]
   \rect{(4,0),(8,5)}
\endmfpic
```

Další možností vyplňování uzavřené křivky je její „pokrytí“ námi nadefinovaným vzorem, jakési „kachličkování“. Vzor vytvoříme pomocí okolí

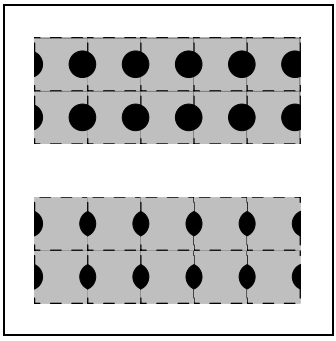
\tile {*jméno_vzoru*, *jednotka*, *šířka*, *výška*, *ořezání*}
...
\endtile ,
\begin{tile} ... \end{tile} (v LaTeXu).

Vzor je tvořen v obdélníku, respektive čtverci, s protilehlými vrcholy $(0,0)$ a (*šířka*, *výška*) a definují ho veškeré příkazy v okolí \tile ... \endtile, respektive begin{tile} ... end{tile}. Povinný parametr *jednotka* určuje jednotku, ve které je daný vzor kreslen; parametr *ořezání* má hodnoty true (kachlička je oříznuta na námi zadaný obdélník nebo čtverec) a false (kachličky nejsou ořezány; při pokládání jsou použity takové, jaké jsme je vytvořili, nezávisle na možné přesahy přes dané rozměry).

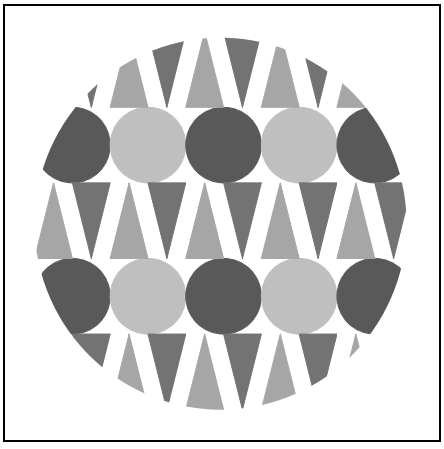
Příkaz

\tess {*jméno_vzoru*} ...

vyplní uzavřenou křivku vzorem *jméno_vzoru*. Vždy musíme použít nějaký námi vytvořený vzor, neboť v mfpic není žádný předdefinovaný. Při pokusu o pokrytí otevřené křivky nedostaneme chybové hlášení, ale křivka bude pouze vykreslena. Kachličkování je prováděno tak, aby levý dolní roh některé z pokrývajících kachliček ležel v počátku, a dělá se postupně zdola nahoru a zleva doprava (viz efekt v případě false na následujícím obrázku).



```
\mfpic[20]{0}{5}{0}{5}
\begin{tile}{kachl_i,20pt,1,1,true}
\dashed\shade\rect{(0,0),(1,1)}
\gfill\circle{(-0.1,0.5),0.25}
\gfill\circle{(1.1,0.5),0.25}
\end{tile}
\tile{kachl_ii,20pt,1,1,false}
\dashed\shade\rect{(0,0),(1,1)}
\gfill\circle{(-0.1,0.5),0.25}
\gfill\circle{(1.1,0.5),0.25}
\endtile
\tess{kachl_i}\rect{(0,0),(5,2)}
\tess{kachl_ii}\rect{(0,3),(5,5)}
\endmfpic
```



```
\mfpic[10]{-7}{7}{-7}{7}
\tile{kachlicka,cm,2,2,true}
\gfill[.35white]
        \circle{(.5,1.5),.5}
\gfill[.75white]
       \circle{(1.5,1.5),.5}
\gfill[.65white]\polygon{
       (0,0),(.25,1),(.5,0)}
\gfill[.45white]\polygon{
       (.5,1),(.75,0),(1,1)}
\gfill[.65white]\polygon{
     (1,0),(1.25,1),(1.5,0)}
\gfill[.45white]\polygon{
     (1.5,1),(1.75,0),(2,1)}
\endtile
\tess{kachlicka}
            \circle{(0,0),7}
\endmfpic
```

2.9.3 Šipky

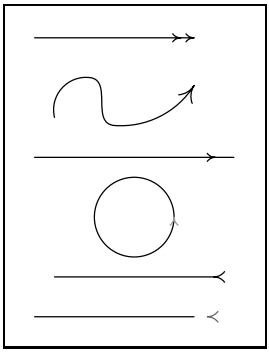
Makro

\arrow [l *délka_šipky*] [r *otočení*] [b *posunutí_zpět*] [c *barva*] .

Jednotlivé nepovinné parametry mají následující význam:

- `b` — posun šipky směrem k počátku křivky, a to po tečně křivky v jejím posledním bodu; přednastavená hodnota pro tento parametr je 0 pt,
- `c` — barva šipky; předdefinovaná barva odpovídá barvě `headcolor`,
- `l` — délka šipky; předdefinovaná délka koresponduje s hodnotou `\headlen`, což je 3 pt,
- `r` — otočení šipky proti směru hodinových ručiček; přednastaven je úhel 0°.

Parametry mohou být zapsány v jakémkoli pořadí. Při použití otočení současně s posunem je třeba myslet na to, že je nejdříve provedeno otočení a poté posun, ovšem ten už probíhá na otočené tečně. Při zápisu nepovinných parametrů tohoto makra může (a nemusí) být mezi `l` a *délkou_šipky*, respektive mezi `r` a *otočením* a podobně, mezera.



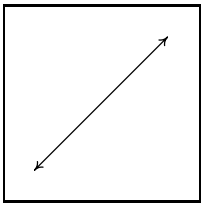
```
\mfpic[15]{0}{5}{0}{7}
\arrow\arrow[b 5pt]\lines{
                          (0,7),(4,7)}
\arrow[l 6pt]\curve{(0.5,5),(1.4,6),
                      (2,4.8),(4,5.8)}
\arrow[b 7pt]\lines{(0,4),(5,4)}
\arrow[c .6white]\circle{
                          (2.5,2.5),1}
\arrow[r180][l4pt]
              \lines{(0.5,1),(4.5,1)}
\arrow[r180][l4pt][b5pt][c.4white]
                   \lines{(0,0),(4,0)}
\endmfpic
```

2.9.4 Změna orientace křivky

Orientaci křivky můžeme jednoduše změnit příkazem

`\reverse` ... .

Tohoto makra využijeme například při vykreslování šipek v počátečním i koncovém bodu křivky:



```
\mfpic[10]{0}{5}{0}{5}
\arrow\reverse\arrow\lines{(0,0),(5,5)}
\endmfpic
```

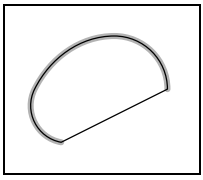
První šipka zleva ve zdrojovém zápisu je kreslena v počátečním bodu dané křivky.

2.9.5 Uzavírání křivek

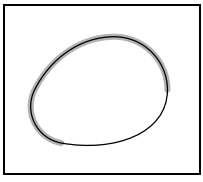
V `mfpic` jsou nadefinované tyto uzavřené křivky: `\rect`, `\circle`, `\ellipse`, `\sector`, `\cyclic`, `\polygon`, `\plrregion` a `\btwnfcn`. Libovolné jiné křivky, které se nám na pohled mohou jevit uzavřenými, `mfpic` chápe jako otevřené.

Pokud jsme vytvořili otevřenou křivku, můžeme využít některý z následujících příkazů pro její uzavření. Všechny uvedené příkazy způsobí uzavření i z pohledu `mfpic`.

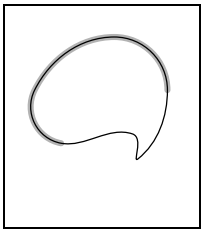
- `\lclosed` — uzavře křivku úsečkou z počátečního do koncového bodu křivky
- `\bclosed` — uzavře křivku Bézierovou křivkou vedenou z prvního k poslednímu bodu dané křivky
- `\cbclosed` — uzavře křivku doplněním kubického B-splinu mezi prvním a posledním bodem zadané křivky
- `\sclosed` — uzavře křivku tak, aby se jevila co nejhladší přičemž v tomto smyslu může být změněn tvar křivky, kterou uzavíráme,
- `\uclosed` — uzavře křivku tak, aby se jevila co nejhladší, ale tvar zadané křivky nezmění



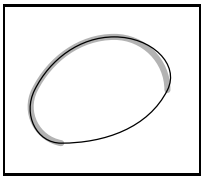
```
\mfpic[20]{0.5}{3}{1}{3}
\pen{2.3pt}
\draw[.7white]\curve{(1,1),(.5,2),
                          (2,3),(3,2)}
\pen{.5pt}
\lclosed\curve{(1,1),(.5,2),(2,3),
                               (3,2)}
\endmfpic
```



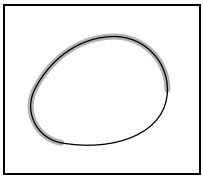
```
\mfpic[20]{0.5}{3}{1}{3}
\pen{2.3pt}
\draw[.7white]\curve{(1,1),(.5,2),
                          (2,3),(3,2)}
\pen{.5pt}
\bclosed\curve{(1,1),(.5,2),(2,3),
                               (3,2)}
\endmfpic
```



```
\mfpic[20]{0.5}{3}{0}{3}
\pen{2.3pt}
\draw[.7white]\curve{(1,1),(.5,2),
                          (2,3),(3,2)}
\pen{.5pt}
\cbclosed\curve{(1,1),(.5,2),(2,3),
                                (3,2)}
\endmfpic
```



```
\mfpic[20]{0.5}{3}{1}{3}
\pen{2.3pt}
\draw[.7white]\curve{(1,1),(.5,2),
                          (2,3),(3,2)}
\pen{.5pt}
\sclosed\curve{(1,1),(.5,2),(2,3),
                                (3,2)}
\endmfpic
```



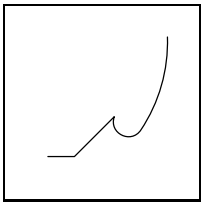
```
\mfpic[20]{0.5}{3}{1}{3}
\pen{2.3pt}
\draw[.7white]\curve{(1,1),(.5,2),
                          (2,3),(3,2)}
\pen{.5pt}
\uclosed\curve{(1,1),(.5,2),(2,3),
                                (3,2)}
\endmfpic
```

2.9.6 Napojení křivek

Okolí

\connect ... \endconnect ,
\begin{connect} ... \end{connect} (pro LaTeX)

sestrojí úsečku z koncového bodu výše zapsané otevřené křivky k počátečnímu bodu další otevřené křivky. Výsledkem je otevřená křivka.



```
\mfpic[10]{0}{5}{0}{5}
\begin{connect}
\lines{(0.5,0.5),(1.5,0.5)}
\curve{(3,2),(4,1.5),(5,5)}
\end{connect}
\endmfpic
```

2.9.7 Afinní transformace

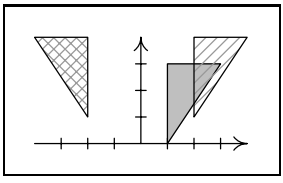
V `mfpic` můžeme použít následující afinní transformace:

- `\rotate` $\{\theta\}$ — otočení o θ stupňů kolem počátku proti směru hodinových ručiček,
- `\rotatearound` $\{(x,y)\}$ $\{\theta\}$ — otočení o θ stupňů kolem bodu (x,y) proti směru hodinových ručiček,
- `\turn` $[(x,y)]$ $\{\theta\}$ — otočení o θ stupňů kolem bodu (x,y) proti směru hodinových ručiček; není-li uveden nepovinný parametr, otáčení je provedeno kolem počátku,
- `\mirror` $\{(x_1,y_1)\}$ $\{(x_2,y_2)\}$ — osová souměrnost zadaná přímkou procházející body (x_1,y_1) a (x_2,y_2),
- `\reflectabout` $\{(x_1,y_1)\}$ $\{x_2,y_2\}$ — osová souměrnost daná přímkou procházející body (x_1,y_1) a (x_2,y_2),
- `\shift` $\{(u,v)\}$ — posunutí o vektor (u,v),
- `\scale` $\{k\}$ — stejnolehlost se středem v počátku a koeficientem k,
- `\xscale` $\{k\}$ — změna měřítka na ose x s koeficientem k,
- `\yscale` $\{k\}$ — změna měřítka na ose y s koeficientem k,
- `\zscale` $\{(u,v)\}$ — stejnolehlost s koeficientem o velikosti vektoru (u,v) a otočení proti směru hodinových ručiček o úhel, který svírá vektor (u,v) s kladnou částí osy x (bodu (x,y) odpovídá bod $(xu-yv, xv+yu)$),
- `\xslant` $\{k\}$ — zkosení ve směru osy x s koeficientem k (obrazem bodu (x,y) je bod $(x+ky,y)$),
- `\yslant` $\{k\}$ — zkosení ve směru osy y s koeficientem k (bodu (x,y) odpovídá bod $(x,kx+y)$),
- `\zslant` $\{(u,v)\}$ — transformace, která bod (x,y) zobrazí na bod $(xu+yv, xv+yu)$,
- `\boost` $\{\chi\}$ — má stejný efekt jako `zslant` $\{(\cosh\chi, \sinh\chi)\}$,
- `\xyswap` — osová souměrnost daná osou $y=x$.

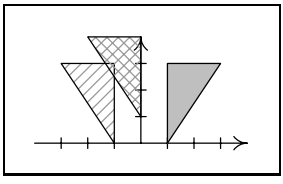
2.9.8 Používání afinních transformací

Použijeme-li libovolnou afinní transformaci, budou se jí podrobovat všechny objekty v aktuálním okolí `\mfpic` ... `\endmfpic` zapsané níže od této transformace. Při aplikaci další transformace dojde ke složení s předcházející.

Dvěma nepříliš složitými obrázky můžeme ukázat, že grupa afinních transformací s operací skládání není komutativní. (U obou obrázků využíváme posunutí o vektor $(1, 1)$ a osovou souměrnost podle osy y, přičemž v prvním obrázku musí být zadaná osa y posunuta o odpovídající vektor, neboť `mfpic` ji před provedením osové souměrnosti posune o vektor předcházejícího posunutí; u druhého obrázku zase upravíme příslušným způsobem posunutí tak, abychom neprováděli posunutí o vektor $(1, 1)$ zobrazený v předchozí osové souměrnosti.)



```
\mfpic[10]{-4}{4}{0}{4}
\axes
\xmarks{-3,-2,-1,1,2,3}
\ymarks{1,2,3}
\store{trojuhelnik}
        {\polygon{(1,0),(1,3),(3,3)}}
\draw\shade\mfobj{trojuhelnik}
\shift{(1,1)}\draw
             \rhatch[\the\hatchspace]
         [.6white]\mfobj{trojuhelnik}
\reflectabout{(-1,-1)}{(-1,3)}\draw
                   \hatch[3][.6white]
                  \mfobj{trojuhelnik}
\endmfpic
```



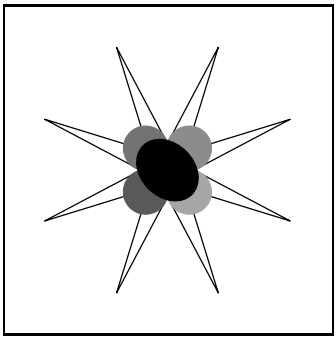
```
\mfpic[10]{-4}{4}{0}{4}
\axes
\xmarks{-3,-2,-1,1,2,3}
\ymarks{1,2,3}
\store{trojuhelnik}
  {\polygon{(1,0),(1,3),(3,3)}}
\draw\shade\mfobj{trojuhelnik}
\reflectabout{(0,0)}{(0,3)}\draw
   \rhatch[\the\hatchspace][.6white]
                  \mfobj{trojuhelnik}
\shift{(-1,1)}\draw
                   \hatch[3][.6white]
                  \mfobj{trojuhelnik}
\endmfpic
```

Pro případ, kdy nám skládání transformací není moc vhod, lze použít okolí

`\coords` ... `\endcoords` ,
`\begin{coords}` ... `\end{coords}` (pro LaTeX).

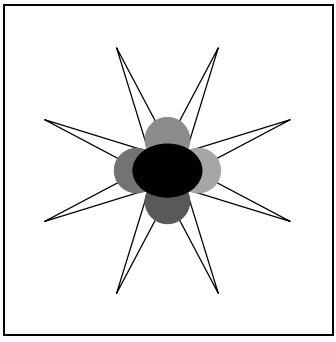
Při otevření okolí dojde k „uschování“ právě aktuální transformace a v okamžiku uzavření je transformace navrácena.

Následující tři obrázky přibližují chování tohoto okolí. První z nich provede postupné otáčení trojúhelníku o 45°, otevřením okolí `coords` je uloženo otočení o $7 \cdot 45°$ a poté je vykreslen (nezapomeňme, že je nejprve otočen o $7 \cdot 45°$) a otočen další útvar.



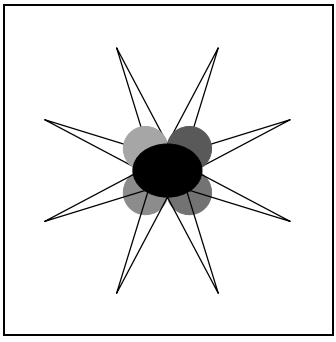
```
\mfpic[10]{-5}{5}{-5}{5}
\store{listik}{\polyline{(1,0),
          5*dir(22.5).dir(45)}}
\store{listecek}{\lclosed\curve{
        dir(-45),(2,0),dir(45)}}
\mfobj{listik}
\rotate{45}\mfobj{listik}
\rotate{45}\mfobj{listik}
\rotate{45}\mfobj{listik}
\rotate{45}\mfobj{listik}
\rotate{45}\mfobj{listik}
\rotate{45}\mfobj{listik}
\rotate{45}\mfobj{listik}
\begin{coords}
\gfill[.65white]\mfobj{listecek}
\rotate{90}\gfill[.55white]
                \mfobj{listecek}
\rotate{90}\gfill[.45white]
                \mfobj{listecek}
\rotate{90}\gfill[.35white]
                \mfobj{listecek}
\end{coords}
\gfill\ellipse{(0,0),1.3,1}
\endmfpic
```

Ve druhém obrázku je na počátku uložena identita, neboť žádná jiná transformace nebyla použita, potom je aplikováno otáčení vždy o 45°, to jest celkem o $7 \cdot 45°$. Uzavřením prvního okolí `coords` je zapomenuto otočení o $7 \cdot 45°$ a je navrácena identita, což způsobí, že útvar vytvořený v dalším okolí `coords` se nijak netransformuje.



```
\mfpic[10]{-5}{5}{-5}{5}
\store{listik}{\polyline{(1,0),
          5*dir(22.5),dir(45)}}
\store{listecek}{\lclosed\curve{
        dir(-45),(2,0),dir(45)}}
\mfobj{listik}
\begin{coords}
\rotate{45}\mfobj{listik}
\rotate{45}\mfobj{listik}
\rotate{45}\mfobj{listik}
\rotate{45}\mfobj{listik}
\rotate{45}\mfobj{listik}
\rotate{45}\mfobj{listik}
\rotate{45}\mfobj{listik}
\end{coords}
\begin{coords}
\gfill[.65white]\mfobj{listecek}
\rotate{90}\gfill[.55white]
                \mfobj{listecek}
\rotate{90}\gfill[.45white]
                \mfobj{listecek}
\rotate{90}\gfill[.35white]
                \mfobj{listecek}
\end{coords}
\gfill\ellipse{(0,0),1.3,1}
\endmfpic
```

Ve třetím obrázku jsou dvě okolí `coords` vnořena do sebe, což je praktické v situacích, kdy využíváme některé z transformací a na určité objekty potřebujeme aplikovat ještě nějaké transformace další. V naší ukázce je tedy trojúhelník nejprve otočen třikrát, ve vnořeném okolí `coords` je první vykreslovaný útvar otočen o $3 \cdot 45°$, druhý útvar je otočen o $3 \cdot 45° + 90°$ atd.; při uzavírání vnořeného okolí `coords` je vrácena hodnota otočení na $3 \cdot 45°$.



```
\mfpic[10]{-5}{5}{-5}{5}
\store{listik}{\polyline{(1,0),
          5*dir(22.5),dir(45)}}
\store{listecek}{\lclosed\curve{
        dir(-45),(2,0),dir(45)}}
\mfobj{listik}
\begin{coords}
\rotate{45}\mfobj{listik}
\rotate{45}\mfobj{listik}
\rotate{45}\mfobj{listik}
\begin{coords}
\gfill[.65white]\mfobj{listecek}
\rotate{90}\gfill[.55white]
                \mfobj{listecek}
\rotate{90}\gfill[.45white]
                \mfobj{listecek}
\rotate{90}\gfill[.35white]
                \mfobj{listecek}
\end{coords}
\rotate{45}\mfobj{listik}
\rotate{45}\mfobj{listik}
\rotate{45}\mfobj{listik}
\rotate{45}\mfobj{listik}
\end{coords}
\gfill\ellipse{(0,0),1.3,1}
\endmfpic
```

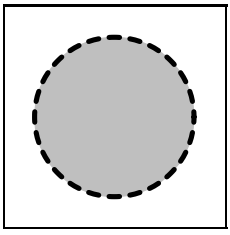
Zdrojové texty předchozích příkladů bychom mohli učiniti trochu „elegantnějšími“ využitím cyklu TEXu, kterým nahradíme sedmkrát prováděné otáčení:

```
\newcount\pocet
\def\kresli#1{\pocet=#1
    \loop\ifnum\pocet>0
         \rotate{45}\mfobj{listik}
         \advance \pocet by -1
    \repeat}
\kresli{7}
```

2.9.9 Pořadí prefixových maker

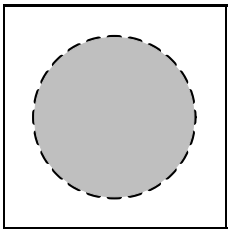
Při práci s tímto druhem maker je důležité si uvědomit, že příkaz vlevo je vždy aplikován na výsledek příkazu vpravo. Jinak řečeno, námi napsaný postup tvorby obrázku je prováděn od konce (zprava).

Podívejme se, jak se mění výsledek kreslení, použijeme-li různá pořadí příkazů \gfill, \dashed:



```
\mfpic[30]{-1}{1}{-1}{1}
\pen{2pt}
\dashed\gfill[0.75white]\circle{(0,0),1}
\endmfpic
```

Nejdříve je nadefinována kružnice se středem $(0, 0)$ a poloměrem 1, poté je vyplněna. Hranice vytvořeného kruhu je vyznačen čárkovanou čarou, přičemž střed pera je veden po hraniční křivce (kružnici), což způsobí, že polovina tloušťky kružnice překrývá výplň kruhu.



```
\mfpic[30]{-1}{1}{-1}{1}
\pen{2pt}
\gfill[0.75white]\dashed\circle{(0,0),1}
\endmfpic
```

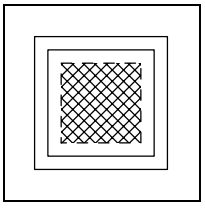
Této posloupnosti příkazů odpovídá překrytí poloviny tloušťky pera, vyznačujícího kružnici, výplní kruhu.

2.10 Rendrování

Redrováním rozumíme proces vlastního překreslení geometrického popisu obrázku. V METAPOSTu je to konkrétně tvorba postscriptového popisu křivek, výplní a podobně. Příkazem

\setrender {*příkazy_TEXu*}

změníme přednastavený způsob rendrování — pomocí \draw.



```
\mfpic[10]{0}{5}{0}{5}
\rect{(0,0),(5,5)}   % \draw není nutné
\draw\rect{(.5,.5),(4.5,4.5)}
\setrender{\dashed\xhatch}
\rect{(1,1),(4,4)}
\endmfpic
```

2.11 Funkce

Makro

\fdef {*název_funkce*} {*proměnná*$_1$, *proměnná*$_2$, ...} {*předpis*}

nadefinuje funkci *název_funkce* (*název_funkce* smí být složen pouze z písmen a podtržítka) o proměnných *proměnná*$_1$, *proměnná*$_2$, ... a předpisem *předpis*, který je předán METAPOSTu ke zpracování. Můžeme tudíž používat všechny operace, které umožňuje METAPOST, tedy $+$, $-$, $*$, $/$, $**$; závorky pro upřesňování pořadí operací (,); znaménka rovnosti, respektive nerovnosti $=$, $<$, $>$, $<=$, $>=$ a další funkce uvedené v následující tabulce:

`round`	zaokrouhlení podle obvyklých pravidel,
`floor`	zaokrouhlení směrem dolů,
`ceiling`	zaokrouhlení směrem nahoru,
`min`	minimum (dva a více argumentů),
`max`	maximum (dva a více argumentů),
`abs`	absolutní hodnota,
`sqrt`	druhá odmocnina,
`sind`	funkce sinus (argument zadáváme ve stupních),
`cosd`	funkce kosinus (argument zadáváme ve stupních),
`mlog`	logaritmus se základem $e^{1/256}$ (`mlog`$= 256 * \ln x$),
`mexp`	inversní funkce k funkci předchozí.

Dále můžeme využít funkce předdefinované v `grafbase.mp`, což jsou tyto:

`tand`	funkce tangens (argument zadáváme ve stupních),
`cotd`	funkce kotangens (argument zadáváme ve stupních),
`secd`	funkce sekans (argument zadáváme ve stupních),
`cscd`	funkce kosekans (argument zadáváme ve stupních),
`asin`	funkce arkussinus (výsledek dostáváme ve stupních),
`acos`	funkce arkuskosinus (výsledek dostáváme ve stupních),
`atan`	funkce arkustangens (výsledek dostáváme ve stupních),
`sin`	funkce sinus (argument zadáváme v radiánech),
`cos`	funkce kosinus (argument zadáváme v radiánech),
`tan`	funkce tangens (argument zadáváme v radiánech),
`cot`	funkce kotangens (argument zadáváme v radiánech),
`sec`	funkce sekans (argument zadáváme v radiánech),
`csc`	funkce kosekans (argument zadáváme v radiánech),
`invsin`	funkce arkussinus (výsledek dostáváme v radiánech),
`invcos`	funkce arkuskosinus (výsledek dostáváme v radiánech),
`invtan`	funkce arkustangens (výsledek dostáváme v radiánech),

`exp`	exponenciální funkce e^x,
`sinh`	funkce hyperbolický sinus,
`cosh`	funkce hyperbolický kosinus,
`tanh`	funkce hyperbolický tangens,
`ln`	přirozený logaritmus,
`asinh`	funkce argument hyperbolického sinu,
`acosh`	funkce argument hyperbolického kosinu,
`atanh`	funkce argument hyperbolického tangens.

Následující čtyři makra mají společný nepovinný parametr *typ_křivky*, který definuje, zda je funkce vykreslena jako Bézierova křivka (odpovídá hodnota parametru `s`) nebo jako lomená čára (odpovídá hodnota `p`). Dále je pro nás podstatné, že smíme využívat pouze proměnných x a t, jak je udáno u jednotlivých maker:

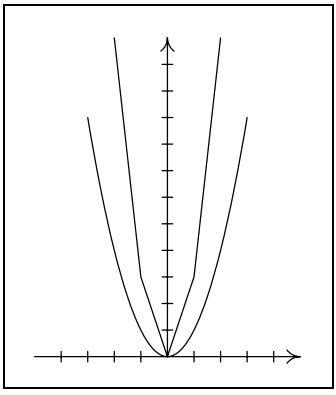
- `\function` [*typ_křivky*] {$x_{\min}$, $x_{\max}$, *krok*} {$f(x)$} — vykresluje funkci $f(x)$ na intervalu $\langle x_{\min}, x_{\max}\rangle$ (přičemž METAPOST postupuje od $x_{\min}$ postupně po kroku *krok* až k $x_{\max}$); nepovinný parametr je přednastaven na `s`,

- `\parafcn` [*typ_křivky*] {$t_{\min}$, $t_{\max}$, *krok*} {$f(t)$} — vytvoří parametricky zadanou křivku $f(t) = (x(t), y(t))$ s parametrem t z intervalu $\langle t_{\min}, t_{\max}\rangle$ (a krokem *krok*); nepovinný argument je přednastaven na `s`,

- `\plrfcn` [*typ_křivky*] {$t_{\min}$, $t_{\max}$, *krok*} {$f(t)$} — určuje křivku $f(t)$ danou polárními souřadnicemi (t, r), kde $r = f(t)$; úhel t nabývá hodnot z intervalu $\langle t_{\min}, t_{\max}\rangle$ a je měřen ve stupních; nepovinný parametr je nastaven jako `s`,

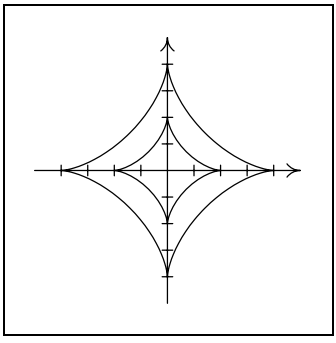
- `\btwnfcn` [*typ_křivky*] {$x_{\min}$, $x_{\max}$, *krok*} {$f(x)$} {$g(x)$} — vyznačí oblast ohraničenou funkcemi $f(x)$, $g(x)$ na intervalu $\langle x_{\min}, y_{\max}\rangle$ a dvěma vertikálními přímkami v $x_{\min}$, $x_{\max}$; parametr *typ_křivky* je nastaven jako `p`,

- `\plrregion` [*typ_křivky*] {$t_{\min}$, $t_{\max}$, *krok*} {$f(t)$} — stanoví oblast ohraničenou křivkou *f(t)* zadanou polárními souřadnicemi (t, r), kde $r = f(t)$; úhel t nabývá hodnot v intervalu $\langle t_{\min}, t_{\max}\rangle$ a je měřen ve stupních; oblast je uzavřena úsečkou spojující počátek souřadné soustavy a bod odpovídající $t_{\min}$ a spojnicí bodu odpovídajícího $t_{\max}$ s počátkem; nepovinný parametr je přednastaven na `p`.



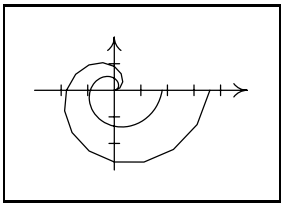
```
\mfpic[10]{5}{5}{0}{12}
\axes
\xmarks{-4,-3,-2,-1,1,2,3,4}
\ymarks{1,2,3,4,5,6,7,8,9,10,11}
\function{-3,3,.5}{x*x}
\function[p]{-2,2,1}{3*x*x}
\endmfpic
```

Asteroida:

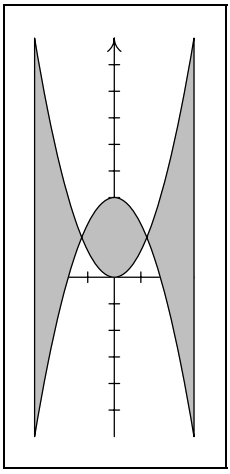


```
\mfpic[10]{-5}{5}{-5}{5}
\axes
\xmarks{-4,-3,-2,-1,1,2,3,4}
\ymarks{-4,-3,-2,-1,1,2,3,4}
\parafcn{0,1440,22.5}{
((3*cosd (t/4))+cosd ((3*t)/4),
(3*sind (t/4))-sind ((3*t)/4))}
\parafcn[p]{0,1440,22.5}{
    (.5*((3*cosd (t/4))+
         cosd ((3*t)/4)),
      .5*((3*sind(t/4))-
        sind ((3*t)/4)))}
\endmfpic
```

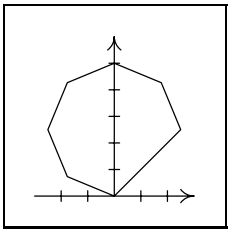
Archimédova spirála:



```
\mfpic[10]{-3}{5}{-3}{2}
\axes
\xmarks{-2,-1,1,2,3,4}
\ymarks{-2,-1,1}
\plrfcn{0,360,22.5}{.005*t}
\plrfcn[p]{0,360,22.5}{.01*t}
\endmfpic
```

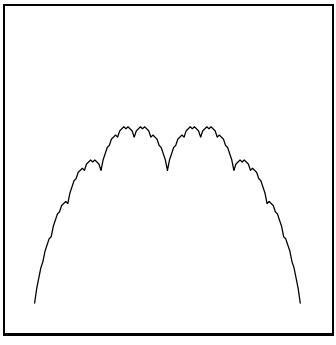


```
\mfpic[10]{-3}{3}{-6}{9}
\axes
\xmarks{-2,-1,1,2}
\ymarks{-5,-4,-3,-2,-1,1,2,3,4,5,6,7,8}
\shade\btwnfcn[s]{-3,3,.5}{x*x}{-x*x+3}
\btwnfcn[s]{-3,3,.5}{x*x}{-x*x+3}
\endmfpic
```



```
\mfpic[10]{-3}{3}{0}{6}
\axes
\xmarks{-2,-1,1,2}
\ymarks{1,2,3,4,5}
\plrregion{45,180,22.5}{5*sind(t)}
\endmfpic
```

Funkce na následujícím obrázku je sedmým částečným součtem řady, která konverguje ke spojité funkci nemající nikde derivaci. V `mfpic` je znázornění takové funkce velmi jednoduché.



```
\mfpic[100]{0}{1}{0}{1}
\fdef{f}{x}{min(x-(floor x),
              (ceiling x)-x)}
\function[p]{0,1,1/128}{f(x)+
                +(.5*f(2*x))+
               +(.25*f(4*x))+
              +(.125*f(8*x))+
             +(.0625*f(16*x))+
            +(.03125*f(32*x))+
           +(.015625*f(64*x))}
\endmfpic
```

2.12 Kreslení dat z externího souboru

Balík maker `mfpic` nám umožňuje vykreslovat útvary sestrojené z dat externích souborů.

Externí soubory použité v tomto textu jsou nazvány `data`, `data1` a podobně. Při spouštění TEXu z instalace emTEX byla tato jména nepostačující (pokud neměla příponu) a musela být doplněna o tečku na konci jména (`data.`).

Příkaz

\datafile {*jméno_souboru*}

vytvoří lomenou čáru procházející body zapsanými v souboru *jméno_souboru*. Předpokládáme, že každá neprázdná řádka obsahuje alespoň dvě čísla. První dvě čísla na řádce reprezentují souřadnice x a y jednoho bodu, ostatní čísla jsou ignorována. Prázdné řádky na začátku souboru jsou ignorovány, všechny další jsou chápány jako konec křivky; komentář je uvozen znakem procenta (%); viz příkaz \mfpdatacomment. Takto vytvořený objekt smíme uzavřít, vybarvit a podobně.

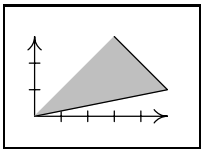
Příkaz

\smoothdata [*napětí*]

způsobí, že výsledkem makra \datafile je Bézierova křivka místo lomené čáry, a příkaz

\unsmoothdata

nastaví zpět použití lomené čáry.



```
\mfpic[10]{0}{5}{0}{3}
\axes
\xmarks{1,2,3,4}
\ymarks{1,2}
\shade\lclosed\datafile{data}
\datafile{data}
\endmfpic
```

Soubor `data` je pro tento příklad zadán:

```
      prázdná řádka
      prázdná řádka
      prázdná řádka
0   0    7          2
5 1         5
3 3

0 0
3 4
0 5
```

Chceme-li uvozovat komentáře jiným způsobem, než-li znakem procenta, nadefinujeme si k tomuto účelu námi zvolený znak příkazem

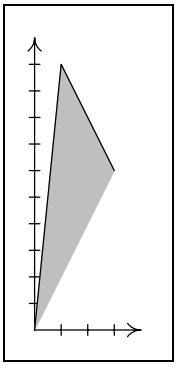
\mfpdatacomment *znak*

Zároveň bude při čtení daného souboru zrušeno zvláštní postavení znaku procenta.

Příkaz

\using {*vstupní_vzor*} {*výstupní_vzor*}

změní standardní způsob zpracování datového souboru. Použijeme-li například vzorek \using {#1 #2 #3}{#3,#1}, bude to znamenat následující: čísla na řádce až po první mezeru jsou přiřazena do parametru #1 *vstupního_vzoru*, další čísla až po druhou mezeru do parametru #2 a zbývající čísla na řádce do parametru #3 *vstupního_vzoru*. Pro vykreslení křivky jsou použity souřadnice dle *výstupního_vzoru* odpovídající jednotlivým parametrům *vstupního_vzoru*. Námi nadefinovaný způsob zpracování datových souborů platí až do konce okolí mfpic, tedy až do příkazu \endmfpic, a vztahuje se na zpracování souborů při použití příkazů \datafile i \plotdata (viz následující příkaz).



```
\mfpic[10]{0}{4}{0}{11}
\axes
\xmarks{1,2,3}
\ymarks{1,2,3,4,5,6,7,8,9,10}
\using{#1 #2 #3}{#2,#1*2}
\shade\lclosed\datafile{data}
\datafile{data}
\endmfpic
```

Pro nakreslení několika lomených čar (nebo Bézierových křivek po využití příkazu \smoothdata) zadaných v jednom datovém souboru do jednoho obrázku je připraveno makro

\plotdata {*jméno_souboru*}

Jednotlivé čáry jsou od sebe odděleny jednou volnou řádkou, více prázdných řádek je chápáno jako konec souboru. Čáry jsou vykreslovány postupně, a to šesti různými typy čar. Příkazem \coloredlines změníme barvu čar (střídá se osm různých barev počínající od černé). Příkazy \pointedlines a \datapointsonly využívají makra \plot a \plotnodes a způsobují vykreslení čar nebo pouze zadaných bodů pomocí devíti různých symbolů známých z makra \plotsymbol (viz strana 34).

Nastavení původního způsobu (to jest různými typy čar) kreslení se provádí příkazem \dashedlines. Chceme-li ovlivnit první typ čáry (respektive barvu nebo druh symbolu), použijeme \mfplinestyle {*číslo*}, kde *číslo* je nezáporné. Typy čar (respektive barvy, druhy symbolu) jsou číslovány od 0; je-li v okolí mfpic více

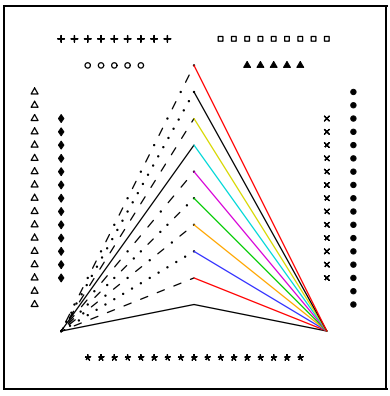
příkazů `\plotdata`, číslování vždy navazuje na číslování předcházející.

U tohoto příkazu se nesmí využívat žádná prefixová makra.

Pro datové soubory `data1`, `data2`, `data3`:

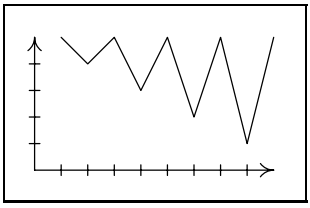
```
data1          data2          data3
 0  0           10  0           1 -1
 5  1            5  1           9 -1

 0  0           10  0          -1  1
 5  2            5  2          -1  9

 0  0           10  0          11  1
 5  3            5  3          11  9

 0  0           10  0           0 11
 5  4            5  4           4 11

 0  0           10  0           6 11
 5  5            5  5          10 11

 0  0           10  0           0  2
 5  6            5  6           0  8

 0  0           10  0          10  2
 5  7            5  7          10  8

 0  0           10  0           1 10
 5  8            5  8           3 10

 0  0           10  0           7 10
 5  9            5  9           9 10

 0  0           10  0
 5 10            5 10
```

dostaneme následující obrázek.



```
\mfpic[10]{-1}{11}{-1}{11}
\plotdata{data1}
\coloredlines
\plotdata{data2}
\pointedlines
\plotdata{data3}
\endmfpic
```

V dalším obrázku využijeme makro \sequence, které reprezentuje posloupnost přirozených čísel. V našem případě nabývají x-ové souřadnice bodů hodnot 1, 2, ... až počet řádků v souboru.



```
\mfpic[10]{0}{9}{0}{5}
\axes
\xmarks{1,2,3,4,5,6,7,8}
\ymarks{1,2,3,4}
\using{#1}{\sequence,#1}
\datafile{data4}
\endmfpic
```

Soubor `data4`:

```
5
4
5
3
5
2
5
1
5
```

2.13 Popisy obrázků

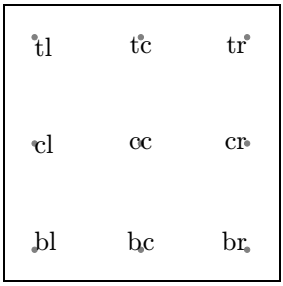
Příkazem

\tlabel [*umístění otočení*] (x, y) {*popiska*}

umístíme k bodu (x, y) text *popiska*. Parametr *umístění* je složen ze dvou písmen; první písmeno udává vertikální polohu (čtyři varianty: `t`, `c`, `b`, `B` odpovídající

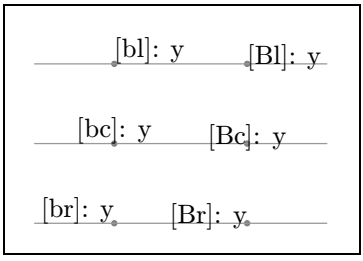
postupně anglickým výrazům top, center, bottom, Baseline), druhé polohu horizontální (tři varianty: `l`, `c`, `r` korespondující s anglickými left, center, right). Mezi písmeny nesmí být mezera. Použitím jednotlivých kombinací získáme popisky umístěné vzhledem k bodu (x, y) podle následujících obrázků; přednastavená hodnota tohoto argumentu je `Bl`.

Všimněme si, že zadáváme polohu bodu vůči popisce (u METAPOSTu tomu bylo naopak).




```
\mfpic[10]{-4}{4}{-4}{4}
\fillcolor{0.5white}
\point{(-4,4),(0,4),(4,4),
       (-4,0),(0,0),(4,0),
    (-4,-4),(0,-4),(4,-4)}
\tlabel[lt](-4,4){lt}
\tlabel[tc](0,4){tc}
\tlabel[tr](4,4){tr}
\tlabel[cl](-4,0){cl}
\tlabel[cc](0,0){cc}
\tlabel[cr](4,0){cr}
\tlabel[bl](-4,-4){bl}
\tlabel[bc](0,-4){bc}
\tlabel[br](4,-4){br}
\endmfpic
```

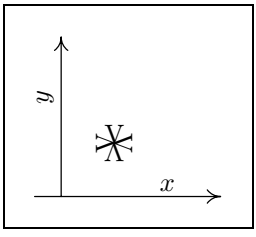
Použijeme-li argument `b` je text sázen tak, aby bounding box písmene byl postaven na účaří, zatímco argument `B` ztotožní y-ovou souřadnici referenčního bodu bounding boxu písmene s y-ovou souřadnicí bodu, který zadáváme. Rozdíl mezi parametrem `b` a `B` je vidět na ukázce písmene y:




```
\mfpic[10]{-3}{8}{-6}{1}
\fillcolor{.5white}
\point{(0,0),(5,0),(0,-3),
     (5,-3),(0,-6),(5,-6)}
\draw[.6white]
      \lines{(-3,0),(8,0)}
\draw[.6white]
    \lines{(-3,-3),(8,-3)}
\draw[.6white]
    \lines{(-3,-6),(8,-6)}
\tlabel[bl](0,0){[bl]: y}
\tlabel[Bl](5,0){[Bl]: y}
\tlabel[bc](0,-3){[bc]: y}
\tlabel[Bc](5,-3){[Bc]: y}
\tlabel[br](0,-6){[br]: y}
\tlabel[Br](5,-6){[Br]: y}
\endmfpic
```

Parametrem *otočení* text otáčíme kolem bodu, který popisujeme; přednastaveno je nula stupňů. Mezi parametry *umístění* a *otočení* může a nemusí být mezera.

O zpracování popisek rozhoduje volba `mplabels` (viz strana 29). Je-li využita, umisťuje a otáčí popisky METAPOST, v opačném případě je otáčení vyloučeno (nepovinný argument *otočení* je ignorován) a text sází TEX.



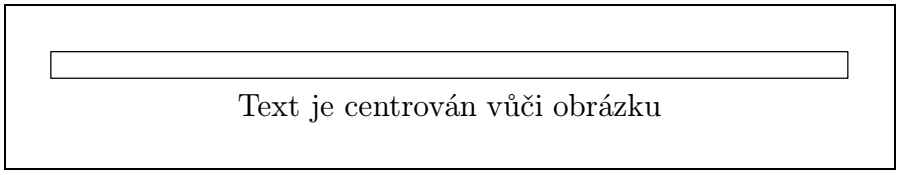

```
\mfpic[20]{-.5}{3}{0}{3}
\axes
\tlabel[bc](2,0.1){$x$}
\tlabel[cr 90](-0.3,2){$y$}
\tlabel[bc](1,1){V}
\tlabel[bc 90](1,1){V}
\tlabel[bc180](1,1){V}
\tlabel[bc 270](1,1){V}
\endmfpic
```

Příkaz

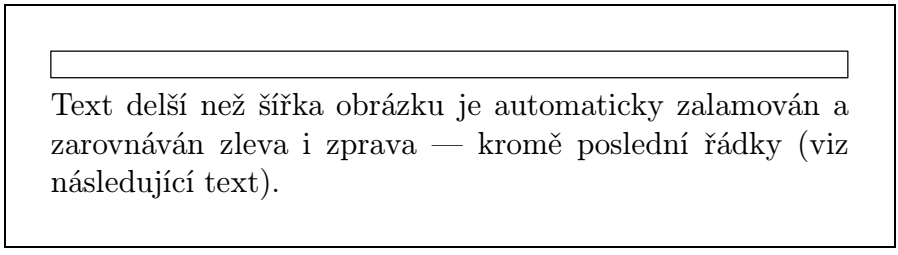
\tcaption [*maximum*, *řádka*] {*text*}

umístí *text* pod obrázek.

Chování makra v různých situacích ukazují nejlépe následující obrázky.

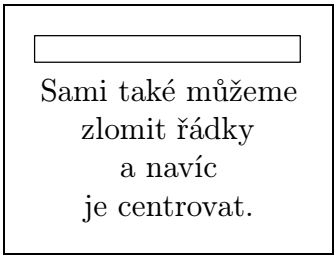



```
\mfpic[10]{0}{30}{0}{1}
\rect{(0,0),(30,1)}
\tcaption{Text je centrován vůči obrázku}
\endmfpic
```




```
\mfpic[10]{0}{30}{0}{1}
\rect{(0,0),(30,1)}
\tcaption{Text delší než šířka obrázku je automaticky zalamován
          a zarovnáván zleva i zprava --- kromě poslední řádky
          (viz následující text).}
\endmfpic
```

Překročí-li délka řádky *textu* hodnotu danou součinem *maxima* a šířky obrázku, jsou řádky lámány tak, aby jejich šířka odpovídala hodnotě součinu *řádky* a šířky obrázku. Přednastavená hodnota *maxima* je 1,2; *řádky* 1,0.



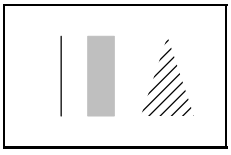

```
\mfpic[10]{0}{10}{0}{1}
\rect{(0,0),(10,1)}
\usecenteredcaptions
\tcaption{Sami také můžeme\\
                zlomit řádky\\
                     a navíc\\
                je centrovat.}
\endmfpic
```

2.14 Pole křivek

Okolí

\patharr {*jméno_pole*} ... \endpatharr ,
\begin{patharr} {*jméno_pole*} ... \end{patharr} (v LaTeXu),

v němž je zapsáno několik křivek, vytvoří z těchto křivek pole nazvané *jméno_pole*. Na jednotlivé položky tohoto pole se odkazujeme příkazy \mfobj{*jméno_pole1*}, \mfobj{*jméno_pole2*}, ...



```
\mfpic[10]{0}{6}{0}{3}
\patharr{pole}
\lines{(1,0),(1,3)}
\rect{(2,0),(3,3)}
\polygon{(4,0),(5,3),(6,0)}
\endpatharr
\mfobj{pole1}
\shade\mfobj{pole2}
\rhatch\mfobj{pole3}
\endmfpic
```

2.15 Definice příkazů

Při otevření nového okolí \mfpic ... \endmfpic jsou vždy znovu nadefinovány všechny příkazy kreslení. Pokud bychom chtěli některý z těchto příkazů předefinovat, musíme to tedy provádět pouze uvnitř okolí \mfpic ... \endmfpic. Ovšem například změna velikosti pera mezi jednotlivými okolími mfpic se v obrázcích projeví.

2.16 Složitější obrázky

Chceme-li tvořit složitější obrázky, budeme zřejmě potřebovat znát kromě příkazů mfpic některé příkazy samotného METAPOSTu a navíc nám mohou posloužit i makra souboru grafbase.mp.

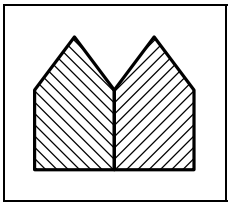
Příkazem

\mfsrc {*příkazy*_METAPOSTu}

zapisujeme *příkazy*_METAPOSTu přímo do výstupního souboru .mp.

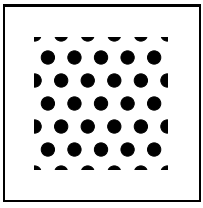
Nadefinujeme-li si vlastní transformaci (přímo v METAPOSTu, musí být v uživatelských souřadnicích), aplikujeme ji pomocí příkazu

\applyT {transformed *transformace*} .



```
\mfpic[10]{-3}{3}{0}{5}
\mfsrc{transform tr;
(1,0) transformed tr = (-1,0);
(2,0) transformed tr = (-2,0);
(1.5,1) transformed tr = (-1.5,1);}
\pen{1.3pt}
\draw\rhatch\polygon{(0,0),(3,0),
                (3,3),(1.5,5),(0,3)}
\draw\lhatch\applyT{transformed tr}
        \polygon{(0,0),(3,0),(3,3),
                        (1.5,5),(0,3)}
\endmfpic
```

Využitím makra `image`, o kterém jsem se zmínila v první části na straně 22, obrázek přímo nevykreslíme, jen ho uložíme do proměnné typu `picture`:



```
\def\uschovej#1#2{%
\mfsrc{picture #1; #1 = image(}%
#2%
\mfsrc{);}}%
\mfpic[10]{0}{5}{0}{5}
\uschovej{mujobr}{
    \polkadot\rect{(0,0),(5,5)}}
\mfsrc{draw mujobr;}
\endmfpic
```

Pokud bychom požadovali vykreslit jen část obrázku „ořízntutou“ uzavřenou křivkou, lze využít makro ze souboru `grafbase.mp`

`clipto` (*proměnná_typu_picture*) *křivka* ,

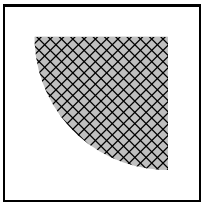
kde *proměnná_typu_picture* je obrázek, který budeme „ořezávat“ křivkou *křivka*. Velmi podobný je příkaz

`clipped` (*proměnná_typu_picture*) *křivka* ,

jen s tím rozdílem, že výstupem je oříznutý obrázek, jenž musí být přiřazen proměnné typu `picture`. *Křivku* je možné vytvořit příkazem `\store` (viz strana 39); je jen vhodné něco vědět o souřadných systémech, se kterými pracujeme.

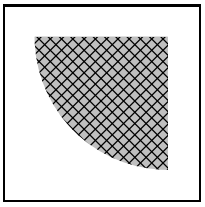
Balík `mfpic` pracuje interně v souřadném systému — device coordinates — odlišném od uživatelského souřadného systému — graph coordinates. Převod z uživatelského systému do interního udává afinní transformace nazvaná `zconv` (inverzní transformací je `invzconv`). Lineární zobrazení indukované touto afinní transformací se nazývá `vconv` (inverzní potom `invvconv`). Např. `\lines{(1,2),(3,-2)}` je ekvivalentní `\mfsrc{draw zconv((1,2)) -- zconv((3,-2));}`.

Pro nás je tedy důležité, že křivka uložená pomocí makra \store je v uživatelských souřadnicích, avšak makra clipto a clipped požadují argument v interních souřadnicích.



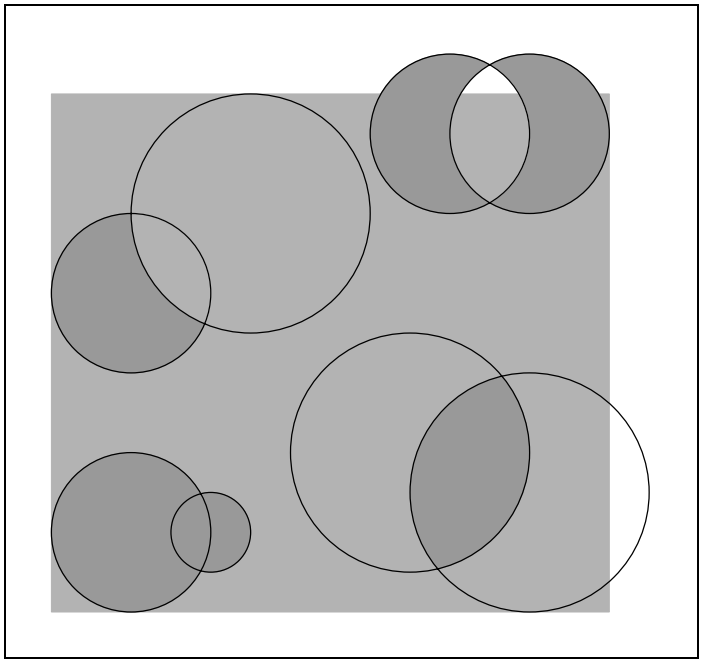
```
\def\uschovej#1#2{%
\mfsrc{picture #1; #1 = image(}%
#2%
\mfsrc{);}}%
\mfpic[10]{0}{5}{0}{5}
\uschovej{dalsiobr}{
\xhatch\shade\rect{(0,0),(5,5)}}
\store{krivka}{\circle{(5,5),5}}
\mfsrc{clipto (dalsiobr)
       zconv(krivka);
       draw dalsiobr;}
\endmfpic
```

Shodný výsledek dává i následující zdrojový text.



```
\def\uschovej#1#2{%
\mfsrc{picture #1; #1 = image(}%
#2%
\mfsrc{);}}%
\mfpic[10]{0}{5}{0}{5}
\uschovej{dalsiobr}{
\xhatch\shade\rect{(0,0),(5,5)}}
\mfsrc{picture o;
         o = clipped (dalsiobr)
          fullcircle scaled 100
               shifted (50,50);
                       draw o;}
\endmfpic
```

Pro kreslení průniku a rozdílu dvou množin také použijeme zmíněná makra. Proměnná active_plane je proměnná METAPOSTu typu picture definovaná v grafbase.mp a ukládá se do ní postupně vše námi nakreslené. Má podobnou úlohu jako v METAPOSTu proměnná currentpicture. Dále si v definici příkazu \rozdil všimněte, že aby šedé pozadí nebylo překryto (barvy v PostScriptu jsou tzv. krycí, tedy pozdější barva překryje předchozí), musela se část pozadí (uloženého v active_plane) „uschovat“ do proměnné pom_iii a na závěr se zase nakreslila. Jinak by zde vznikl „otvor“ v barvě background (ta se používá v příkazu undraw pro mazání).



```
\def\uschovej#1#2{%
 \mfsrc{picture #1; #1 = image(}%
 #2%
 \mfsrc{);}}%
\def\sjednoceni#1#2{%
\gfill#1\gfill#2}%
\def\prunik#1#2{%
 \uschovej{mnozina_i}{\gfill #1}
 \store{krivka_i}{#2}
 \mfsrc{clipto (mnozina_i) zconv(krivka_i);
 draw mnozina_i;}}%
\def\rozdil#1#2{%
 \uschovej{mnozina_i}{\gfill #1}
 \store{krivka_i}{#1}
 \store{krivka_ii}{#2}
 \mfsrc{picture mnozina_ii;
 mnozina_ii=clipped (mnozina_i) zconv(krivka_ii);
 mnozina_ii:=image(draw mnozina_i; undraw mnozina_ii;);}
 \mfsrc{picture mnozina_iii; mnozina_iii=clipped (active_plane)
 zconv(krivka_i); clipto (mnozina_iii) zconv(krivka_ii);}
 \mfsrc{draw mnozina_ii; draw mnozina_iii;}}%
\def\symrozdil#1#2{%
 \rozdil{#1}{#2}
 \rozdil{#2}{#1}}%
\mfpic[30]{0}{7.5}{0}{7}
```

```
\gfill[.7white]\rect{(0,0),(7,6.5)}
\fillcolor{.6white}
\sjednoceni{\circle{(1,1),1}}{\circle{(2,1),.5}}
\circle{(1,1),1}
\circle{(2,1),.5}
\prunik{\circle{(4.5,2),1.5}}{\circle{(6,1.5),1.5}}
\circle{(4.5,2),1.5}
\circle{(6,1.5),1.5}
\rozdil{\circle{(1,4),1}}{\circle{(2.5,5),1.5}}
\circle{(1,4),1}
\circle{(2.5,5),1.5}
\symrozdil{\circle{(6,6),1}}{\circle{(5,6),1}}
\circle{(6,6),1}
\circle{(5,6),1}
\endmfpic
```

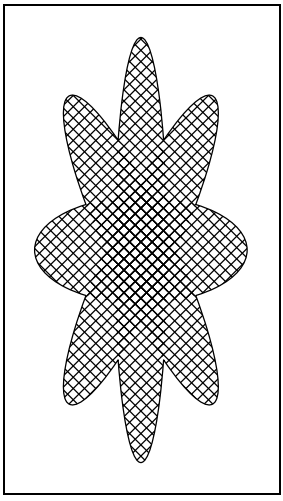
Kopretina na titulní straně této práce také využívá průniky množin. Autorem tohoto obrázku je doc. J. Kuben a zdrojový text vypadá takto (komentářem je barevná varianta):

```
\mfpic[5]{-20}{20}{-20}{20}
\newcommand{\obraz}[2]{%
\mfsrc{picture #1; #1=image(}%
#2%
\mfsrc{);}}
\newcommand{\rozdil}[2]{%
\obraz{pom_i}{\gfill #1}%
\store{krivka_i}{#1}    \store{krivka_ii}{#2}%
\mfsrc{picture pom_ii; pom_ii=clipped (pom_i) zconv(krivka_ii);
pom_ii:=image(draw pom_i; undraw pom_ii;);}%
\mfsrc{picture pom_iii;
pom_iii=clipped (active_plane) zconv(krivka_i);
clipto (pom_iii) zconv(krivka_ii);}%
\mfsrc{draw pom_ii; draw pom_iii;}}
\newcommand{\symrozdil}[2]{\rozdil{#1}{#2}\rozdil{#2}{#1}}
\gfill[.6white]\circle{(0,0),12}  %\gfill[green]\circle{(0,0),12}
\gfill\circle{(0,0),4}            %\gfill[red]\circle{(0,0),4}
\store{ki}{\bclosed\plrfcn{0,360,1.25}{12+8*cosd(8t)}}
\store{kii}{\bclosed\plrfcn{0,360,1.25}{12-8*cosd(8t)}}
\fillcolor{.7white}               %\fillcolor{yellow}
\symrozdil{\mfobj{ki}}{\mfobj{kii}}
%\drawcolor[named]{OrangeRed}
\mfobj{ki}\mfobj{kii}
\endmfpic
```

Křivky v interních souřadnicích jsou argumenty dalšího příkazu — `clipsto`:

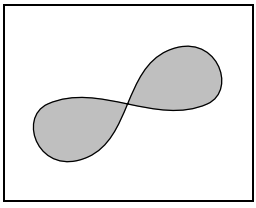
`clipsto` (*proměnná_typu_picture*) (*pole_uzavřených_křivek*) ,

který ořeže *proměnnou_typu_picture* na sjednocení vnitřků všech křivek, které jsou obsaženy v *poli_uzavřených_křivek*.

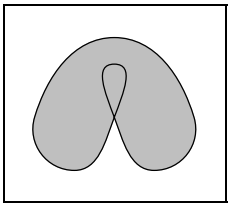


```
\def\uschovej#1#2{%
 \mfsrc{picture #1; #1 = image(}%
 #2%
 \mfsrc{);}}%
\mfpic[10][20]{-4}{4}{-4}{4}
\uschovej{ctverec}{
    \xhatch\rect{(-4,-4),(4,4)}}
\patharr{pole}
\ellipse{(0,0),4,1}
\ellipse[45]{(0,0),4,1}
\ellipse[90]{(0,0),4,1}
\ellipse[135]{(0,0),4,1}
\endpatharr
\mfsrc{for i=1 upto pole:
pole[i]:=zconv(pole[i]); endfor;
         clipsto (ctverec,pole);
                  draw ctverec;}
\mfsrc{path pompole[];
for i=1 upto pole:
pompole[i]:=
(subpath (-.25*length(pole[i]),
.25*length(pole[i])) of pole[i]);
endfor;
for i=1 upto pole:
pompole[i+4]:=
(subpath (.25*length(pole[i]),
.75*length(pole[i])) of pole[i]);
endfor;
draw (buildcycle(pompole1 for i=2
upto 2pole: ,pompole[i] endfor));}
\endmfpic
```

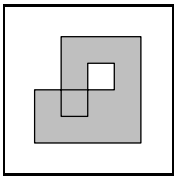
Na závěr se zmíníme o vybarvování uzavřených křivek. PostScript má dva způsoby, jak vyplňuje uzavřené křivky. METAPOST používá ten, který vybarví daný bod, pokud počet oběhů křivky kolem tohoto bodu (tzv. index bodu vzhledem ke křivce) je nenulový. Orientace po směru nebo proti směru hodinových ručiček (tedy zda je index kladný nebo záporný) nehraje roli. Situaci ilustrují následující obrázky.



```
\mfpic[10]{-3.5}{3.5}{-2.5}{2.5}
\draw\gfill[.75white]
\cyclic{(-3,0),(-1.5,-2),(1.5,2),(3,0)}
\endmfpic
```



```
\mfpic[10]{-3}{3}{-2}{3}
\draw\gfill[.75white]
\cyclic{(-3,0),(-1.5,-2),(0,0),
(0,2),(0,0),(1.5,-2),(3,0),(0,3)}
\endmfpic
```



```
\mfpic[10]{0}{4}{0}{4}
\draw\gfill[.75white]
\polygon{(0,0),(4,0),(4,4),(1,4),
(1,1),(2,1),(2,3),(3,3),(3,2),(0,2)}
\endmfpic
```

3 Použití `mfpic` při tvorbě obrázků

V další části mé práce bych ráda na několika obrázcích ze středoškolské tematiky prakticky demonstrovala využití balíku `mfpic`. Pro učitele jsou to většinou obrázky použitelné pro přípravu na vyučování.

Obrázky jsou umístěny záměrně tak, aby je bylo možno porovnávat se zdrojovým kódem.

Ve většině obrázků je použito makro

\mfsrc {*příkazy* METAPOSTu} ,

pomocí něhož vpisujeme do výstupního souboru `.mp` libovolné příkazy METAPOSTu.

U dvou obrázků je ukázáno použití příkazu

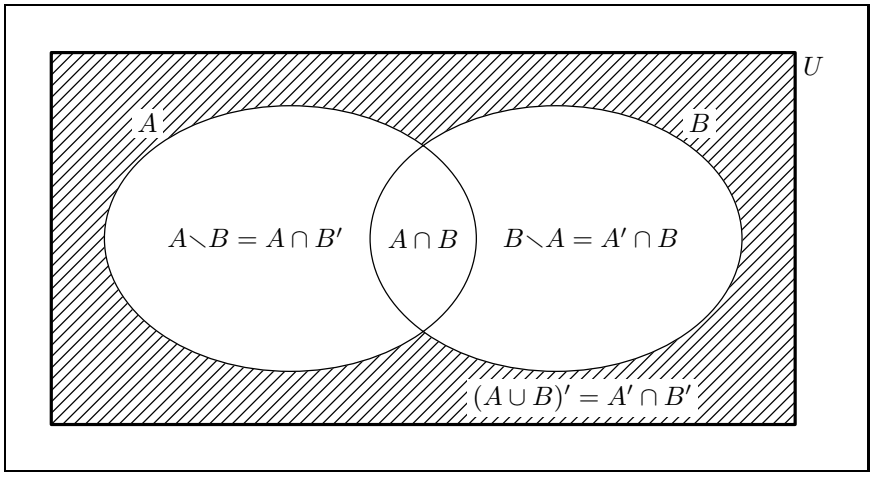
\mfpverbtex {*příkazy* TEXu} ,

který zapíše *příkazy* TEXu ohraničené příkazy `verbatimtex` ... `etex` do výstupního souboru `.mp` (viz strana 15).

3.1 Grafické znázornění množin

Obrázek zachycuje Vennův diagram znázorňující obecnou polohu dvou množin.

Je zde naznačeno využití příkazu `\mfpverbtex`. Pokud by byla skutečně použita jeho zakomentovaná forma, znak rozdílu množin by byl vysázen LaTeXem, což je v tomto případě vhodné. Samozřejmě je ještě třeba upravit soubor `makempx` (viz první část práce na straně 14). Znak rozdílu množin lze také vysázet příkazem `\setminus` (nemusíme nic upravovat, protože plain TeX ho zná), ale výsledek není ideální. Třetí možností je nadefinování vlasního symbolu. Makro `\obd` vytvoří pod popiskou bílý obdélník.



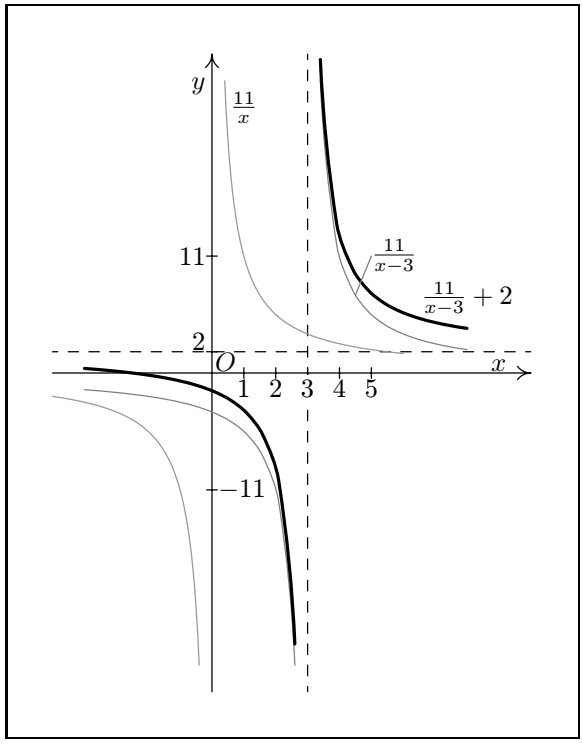
```
%\mfpverbtex{\documentclass{article}
%            \usepackage{amsmath,amsfonts,amssymb}
%            \begin{document}}
\mfpverbtex{%
\def\smallsetminus{{\font\amsb=msbm10
   \mathbin{\hbox{\amsb\char"72}}}}%
\def\obd#1{%
\mfsrc{picture obr; obr = image(}%
\tlabel#1%
\mfsrc{); unfill bbox (obr); draw obr; }%
}%
\mfpic[10]{-14}{15}{-7}{7}
\pen{1.3pt}
\rect{(-14,-7),(14,7)}
\pen{.5pt}
\rhatch\rect{(-14,-7),(14,7)}
\gfill[white]\ellipse{(5,0),7,5}
\draw\gfill[white]\ellipse{(-5,0),7,5}
\ellipse{(5,0),7,5}
\tlabel[cl](14.3,6.5){$U$}
\obd{[br](-10,4){$A$}}
\obd{[bl](10,4){$B$}}
\obd{[cc](6,-6){$(A\cup B)'=A' \cap B'$}}
\tlabel[cc](0,0){$A\cap B$}
%\tlabel[cl](3,0){$B\setminus A=A' \cap B$}
\tlabel[cl](3,0){$B\smallsetminus A=A' \cap B$}
%\tlabel[cr](-3,0){$A\setminus B=A \cap B'$}
\tlabel[cr](-3,0){$A\smallsetminus B=A \cap B'$}
\endmfpic
```

3.2 Grafy funkcí

Graf funkce $g\colon y = \frac{2x+5}{x-3}$.

U tohoto obrázku si znovu ukážeme praktické použití příkazu `\mfpverbtex`; pokud by byl skutečně využit, mohli bychom požadované popisky vysázet příkazem `\frac`.

Makro `\obd` nám zajistí, aby se popiska bodu na ose x nepřekrývala s přerušovanou čarou naznačující posun osy y při konstrukci funkce.

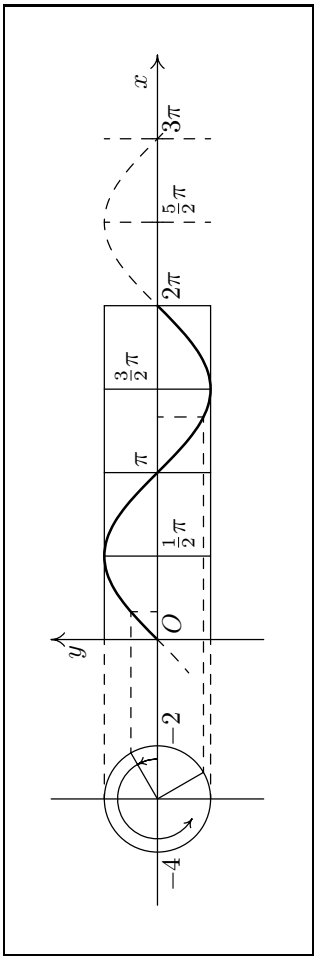



```
%\mfpverbtex{\documentclass{article}
%            \begin{document}}
 %%% makro pro umístění textu na vybarvenou plochu
\def\obd#1{%
\mfsrc{picture obr; obr = image(}%
\tlabel#1%
\mfsrc{); unfill bbox (obr); draw obr; }%
}%
```

```
\mfpic[12][4]{-5}{10}{-30}{30}
\axes
\xmarks{1,2,3,4,5}
\ymarks{-11,2,11}
\tlabel[cr](-.2,27){$y$}
\tlabel[bc](9,.2){$x$}
\tlabel[bl](.1,.2){$O$}
\dashed\lines{(-5,2),(10,2)}
\dashed\lines{(3,-30),(3,30)}
\draw[.6white]\function{-5,-.4,.1}{11/x}
\draw[.6white]\function{.4,6,.1}{11/x}
\draw[.5white]\function{-4,2.6,.5}{11/(x-3)}
\draw[.5white]\function{3.4,8,.5}{11/(x-3)}
\pen{1.3pt}
\function{-4,2.6,.5}{(11/(x-3))+2}
\function{3.4,8,.5}{(11/(x-3))+2}
\pen{.5pt}
\tlabel[cr](-.2,11){11}
\tlabel[cl](.2,-11){$-11$}
\tlabel[br](-.2,2.2){2}
\tlabel[bc](1,-2.3){1}
\tlabel[bc](2,-2.3){2}
\obd{[bc](3,-2.3){3}}
\tlabel[bc](4,-2.3){4}
\tlabel[bc](5,-2.3){5}
\draw[.5white]\lines{(4.5,11/1.5),(5,11)}
%\tlabel[cc](1,25){$\frac{11}{x}$}
%\tlabel[cl](5,11){$\frac{11}{x-3}$}
%\tlabel[cc](8,7){$\frac{11}{x-3}+2$}
\tlabel[cc](1,25){$11\over x$}
\tlabel[cl](5,11){$11\over {x-3}$}
\tlabel[cc](8,7){${11\over {x-3}}+2$}
\endmfpic
```

Graf funkce sinus.

Chceme-li obrázek otočit o devadesát stupňů, lze to provést až po nakreslení celého obrázku, nebo můžeme na úvod zapsat příkaz \rotate{90} a potom ještě otáčet všechny popisky (tento způsob je zakomentovaný a otočení popisek naznačeno u první z nich).

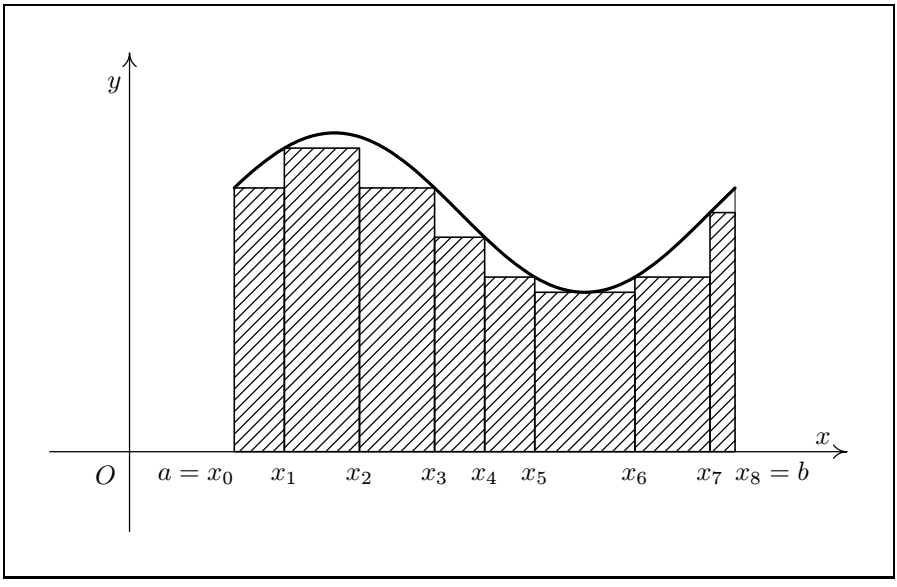



```
\mfpic[20]{-5}{11}{-2}{2}
%\rotate{90}
\axes
\xmarks{(5/2)*pi,3*pi}
%\tlabel[cr 90](-.1,1.5){$y$}
\tlabel[cr](-.1,1.5){$y$}
\tlabel[bc](10.5,.2){$x$}
```

```
\lines{(-3,-2),(-3,2)}
\lines{(0,1),(2*pi,1)}
\lines{(0,-1),(2*pi,-1)}
\lines{(.5*pi,-1),(.5*pi,1)}
\lines{(pi,-1),(pi,1)}
\lines{((3/2)*pi,-1),((3/2)*pi,1)}
\lines{(2*pi,-1),(2*pi,1)}
\dashed\lines{((5/2)*pi,-1),((5/2)*pi,1)}
\dashed\lines{(3*pi,-1),(3*pi,1)}
\fdef{f}{x}{sin x}
\dashed\function{-pi/5,3.1*pi,.1*pi}{f(x)}
\pen{1pt}
\function{0,2*pi,.1*pi}{f(x)}
\pen{.5pt}
\circle{(-3,0),1}
\arrow[r-5]\arc[p]{(-3,0),0,30,.75}
\arrow[r -5]\arc[p]{(-3,0),0,240,.75}
\lines{(-3,0),(-3+cosd 30,sind 30)}
\lines{(-3,0),(-3-cosd 240,sind 240)}
\dashed\lines{(-3,1),(0,1)}
\dashed\lines{(-3,-1),(0,-1)}
\dashed\lines{(-3+cosd 30,sind 30),(pi/6,f(pi/6))}
\dashed\lines{(pi/6,0),(pi/6,f(pi/6))}
\dashed\lines{(-3-cosd 240,sind 240),((4/3)*pi,f((4/3)*pi))}
\dashed\lines{((4/3)*pi,0),((4/3)*pi,f((4/3)*pi))}
\tlabel[tr](-4.1,-.1){$-4$}
\tlabel[tl](-2,-.1){$-2$}
\tlabel[tl](.1,-.1){$0$}
\tlabel[tl]((.5*pi)+.1,-.1){${1\over 2} \pi$}
\tlabel[bl](1*pi+.1,0.2){$\pi$}
\tlabel[bl]((3/2)*pi+.1,.2){${3\over 2} \pi$}
\tlabel[tl](2*pi+.1,-.1){$2\pi$}
\tlabel[tl]((5/2)*pi+.1,-.1){${5\over 2} \pi$}
\tlabel[tl](3*pi+.1,-.1){$3\pi$}
\mfsrc{picture a; a := currentpicture;
          currentpicture:=nullpicture;
                    draw a rotated 90;}%
\endmfpic
```

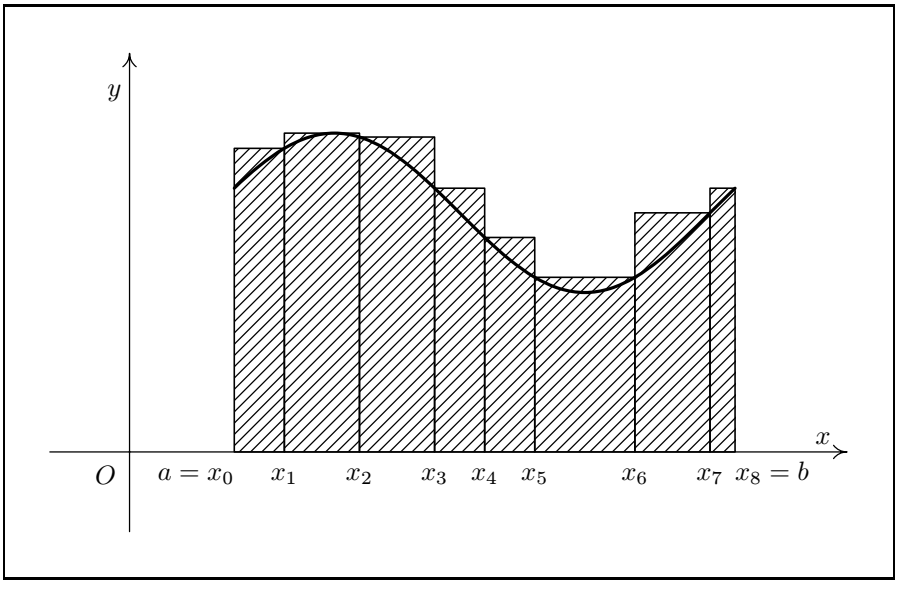
3.3 Určitý integrál a jeho geometrické aplikace

Znázornění dolního integrálního součtu příslušného určitému dělení intervalu $\langle a, b\rangle$.



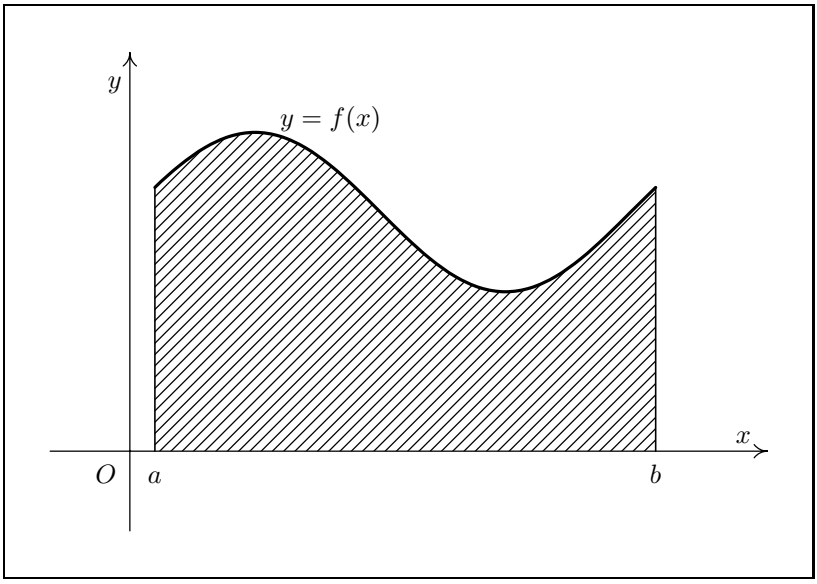
```
\mfpicunit=30pt
\mfpic[1]{-2}{8}{-1}{5}
\xaxis
\headlen=5pt                  %%% musíme změnit přednastavenou
\arrow\lines{(-1,-1),(-1,5)} %%% délku, neboť je jiná než
\tlabel[cr](-1.1,4.6){$y$}   %%% u makra \xaxis; nebo lze
\tlabel[bc](7.7,0.1){$x$}    %%% \arrow[l 5pt]\lines{...}
\tlabel[bc](-1.3,-0.4){$0$}
\fdef{f}{x}{(sin x)+3}
\pen{1.3pt}
\function{.1*pi,2.1*pi,.1*pi}{f(x)}
\pen{.7pt}
\draw\rhatch\rect{(.1*pi,f(.1*pi)),(.3*pi,0)}
\draw\rhatch\rect{(.3*pi,f(.3*pi)),(.6*pi,0)}
\draw\rhatch\rect{(.6*pi,f(.9*pi)),(.9*pi,0)}
\draw\rhatch\rect{(.9*pi,f(1.1*pi)),(1.1*pi,0)}
\draw\rhatch\rect{(1.1*pi,f(1.3*pi)),(1.3*pi,0)}
\draw\rhatch\rect{(1.3*pi,f(1.5*pi)),(1.7*pi,0)}
\draw\rhatch\rect{(1.7*pi,f(1.7*pi)),(2*pi,0)}
\draw\rhatch\rect{(2*pi,f(2*pi)),(2.1*pi,0)}
\pen{.3pt}
\lines{(2.1*pi,f(2*pi)),(2.1*pi,f(2.1*pi))}
\tlabel[br](.1*pi,-0.4){$a=x_0$}
\tlabel[bc](.3*pi,-0.4){$x_1$}
\tlabel[bc](.6*pi,-0.4){$x_2$}
\tlabel[bc](.9*pi,-0.4){$x_3$}
\tlabel[bc](1.1*pi,-0.4){$x_4$}
\tlabel[bc](1.3*pi,-0.4){$x_5$}
\tlabel[bc](1.7*pi,-0.4){$x_6$}
\tlabel[bc](2*pi,-0.4){$x_7$}
\tlabel[bl](2.1*pi,-0.4){$x_8=b$}
\endmfpic
```

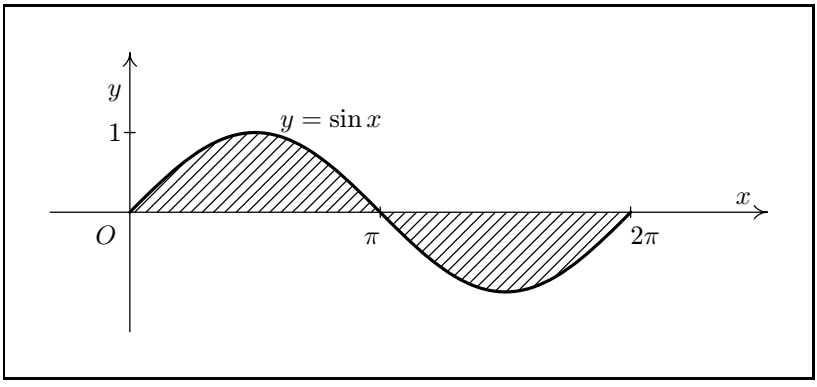
Znázornění horního integrálního součtu příslušného určitému dělení intervalu $\langle a, b\rangle$.



```
\mfpic[1]{-2}{8}{-1}{5}
\mfpicunit=30pt
\xaxis
\headlen=5pt                    %%% pro dosažení stejné délky
\arrow\lines{(-1,-1),(-1,5)} %%% šipky jako v makru \xaxis;
\tlabel[cr](-1.1,4.6){$y$}   %%% nebo lze použít:
\tlabel[bc](7.7,0.1){$x$}    %%% \arrow[l 5pt]\lines{...}
\tlabel[bc](-1.3,-0.4){$0$}
\fdef{f}{x}{(sin x)+3}
\pen{1.3pt}
\function{.1*pi,2.1*pi,.1*pi}{f(x)}
\pen{.7pt}
\draw\rhatch\rect{(.1*pi,f(.3*pi)),(.3*pi,0)}
\draw\rhatch\rect{(.3*pi,f(.5*pi)),(.6*pi,0)}
\draw\rhatch\rect{(.6*pi,f(.6*pi)),(.9*pi,0)}
\draw\rhatch\rect{(.9*pi,f(.9*pi)),(1.1*pi,0)}
\draw\rhatch\rect{(1.1*pi,f(1.1*pi)),(1.3*pi,0)}
\draw\rhatch\rect{(1.3*pi,f(1.7*pi)),(1.7*pi,0)}
\draw\rhatch\rect{(1.7*pi,f(2*pi)),(2*pi,0)}
\draw\rhatch\rect{(2*pi,f(2.1*pi)),(2.1*pi,0)}
\pen{.5pt}
\tlabel[br](.1*pi,-0.4){$a=x_0$}
\tlabel[bc](.3*pi,-0.4){$x_1$}
\tlabel[bc](.6*pi,-0.4){$x_2$}
\tlabel[bc](.9*pi,-0.4){$x_3$}
\tlabel[bc](1.1*pi,-0.4){$x_4$}
\tlabel[bc](1.3*pi,-0.4){$x_5$}
\tlabel[bc](1.7*pi,-0.4){$x_6$}
\tlabel[bc](2*pi,-0.4){$x_7$}
\tlabel[bl](2.1*pi,-0.4){$x_8=b$}
\endmfpic
```

Geometrický význam určitého integrálu $\int_a^b f(x)\,\mathrm{d}x$ funkce f a náčrt obsahu obrazce ohraničeného grafem funkce $f\colon y = \sin x$ v intervalu $\langle 0, 2\pi\rangle$ a osou x.

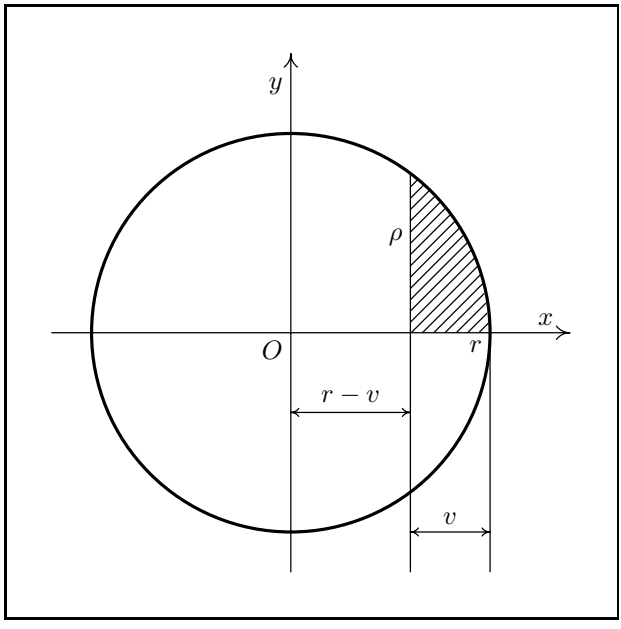




```
\mfpic[1]{-1}{8}{-1}{5}
\mfpicunit=30pt
\axes
\tlabel[cr](-0.1,4.6){$y$}
\tlabel[bc](7.7,0.1){$x$}
\tlabel[bc](-0.1,-0.4){$0$}
\fdef{f}{x}{(sin x)+3}
\pen{1.3pt}
\function{.1*pi,2.1*pi,.1*pi}{f(x)}
\pen{.5pt}
\rhatch\btwnfcn{.1*pi,2.1*pi,.1*pi}{0}{f(x)}
\pen{.7pt}
\lines{(.1*pi,0),(.1*pi,f(.1*pi))}
\lines{(2.1*pi,0),(2.1*pi,f(2.1*pi))}
\tlabel[bl](.6*pi,4){$y=f(x)$}
\tlabel[bc](.1*pi,-0.4){$a$}
\tlabel[bc](2.1*pi,-0.4){$b$}
\endmfpic
```

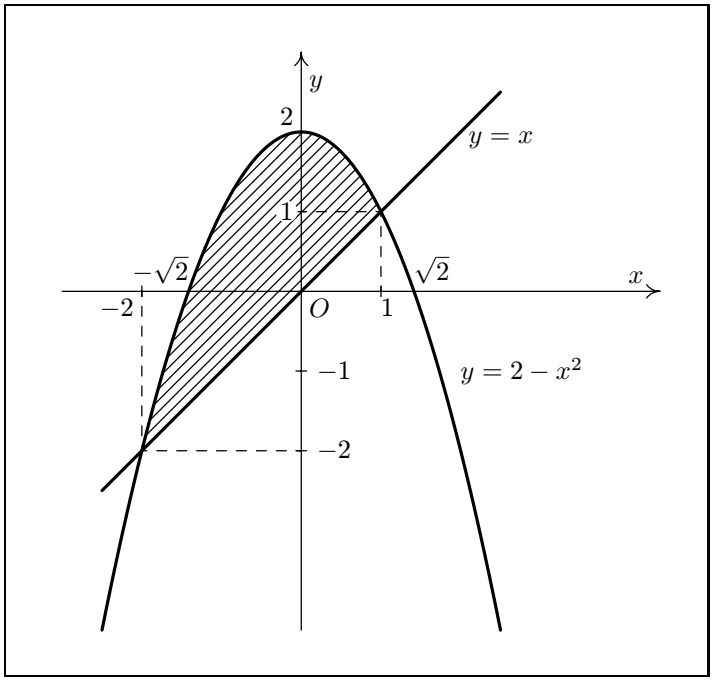
```
\mfpic[1]{-1}{8}{-1.5}{2}
\mfpicunit=30pt
\axes
\ymarks{1}
\xmarks{pi,2*pi}
\tlabel[cr](-0.1,1.5){$y$}
\tlabel[bc](7.7,0.1){$x$}
\tlabel[bc](-.3,-.4){$0$}
\fdef{f}{x}{sin x}
\pen{1.3pt}
\function{0,2*pi,.1*pi}{f(x)}
\pen{.7pt}
\rhatch\btwnfcn{0,2*pi,.1*pi}{0}{f(x)}
\pen{.5pt}
\tlabel[br](pi,-.4){$\pi$}
\tlabel[bl](2*pi,-.4){$2\pi$}
\tlabel[cr](-.1,1){$1$}
\tlabel[bl](.6*pi,1){$y=\sin{x}$}
\endmfpic
```

Náčrt k odvození vzorce pro objem kulové úseče výšky v, která je vyťata z koule o poloměru r.



```
\mfpic[1]{-3}{3.5}{-3}{3.5}
\mfpicunit=30pt
\axes
\tlabel[cr](-.1,3.1){$y$}
\tlabel[bc](3.2,.1){$x$}
\tlabel[tr](-.1,-.1){$0$}
\pen{1.3pt}
\circle{(0,0),2.5}
\pen{.5pt}
\lines{(1.5,sqrt((2.5*2.5)-(1.5*1.5))),(1.5,-3)}
\lines{(2.5,0),(2.5,-3)}
\arrow\reverse\arrow\lines{(0,-1),(1.5,-1)}
\arrow\reverse\arrow\lines{(1.5,-2.5),(2.5,-2.5)}
\pen{.7pt}
\rhatch\btwnfcn{1.5,2.5,.1}{0}{sqrt((2.5*2.5)-(x*x))}
\tlabel[cr](1.4,1.2){$\rho$}
\tlabel[tr](2.4,-.1){$r$}
\tlabel[bc](.75,-.9){$r-v$}
\tlabel[bc](2,-2.4){$v$}
\endmfpic
```

Obsah obrazce ohraničeného grafy funkcí $f\colon y = 2 - x^2$, $g\colon y = x$.

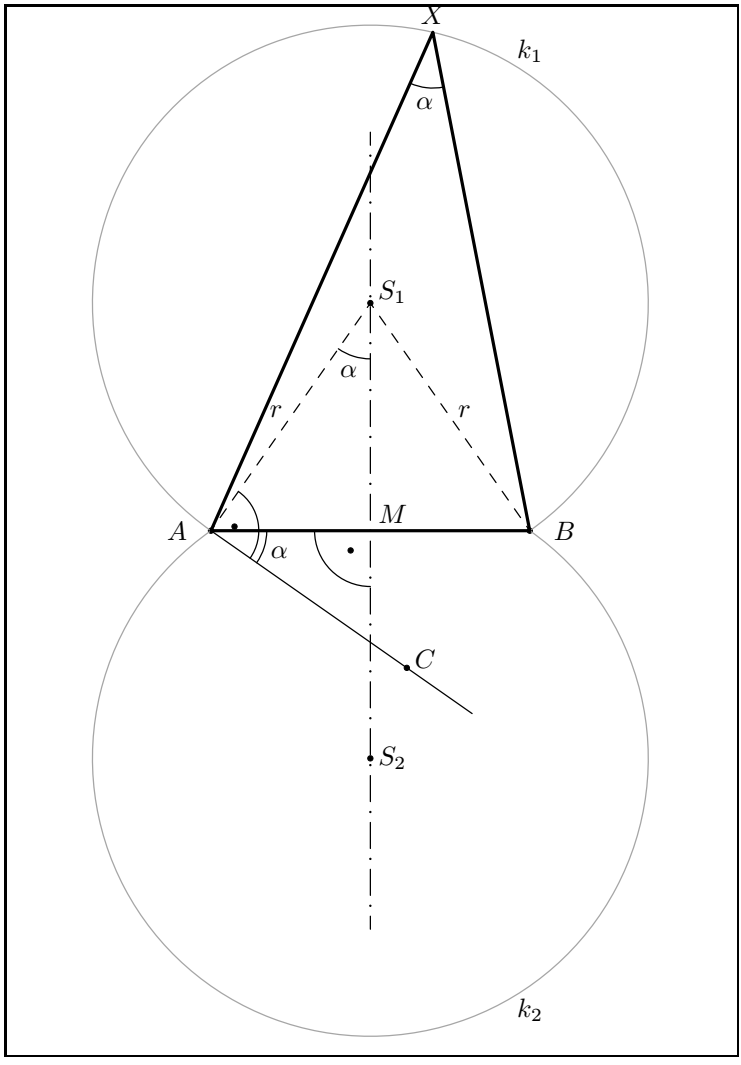


```
\def\obd#1{%                        %%% makro pro odbarvení plochy
\mfsrc{picture obr; obr = image(} %%% pod textem ve tvaru kruhu;
\tlabel#1%                          %%% příkaz center určuje střed
\mfsrc{); unfill fullcircle         %%% bounding boxu
    scaled (2*(abs( center obr -
                  llcorner obr)))
           shifted (center obr);
                     draw obr;}%
\mfpic[1]{-3}{4.5}{-4.25}{3}
\mfpicunit=30pt
\axes
\xmarks{-2,1}
\ymarks{-2,-1,1}
\tlabel[cl](0.1,2.6){$y$}
\tlabel[bc](4.2,.1){$x$}
\tlabel[tl](.1,-.1){$0$}
\fdef{f}{x}{x}
\fdef{g}{x}{2-(x*x)}
\pen{1.3pt}
\function{-2.5,2.5,.1}{f(x)}
\function{-2.5,2.5,.1}{g(x)}
\pen{.7pt}
\rhatch\btwnfcn{-2,1,.1}{g(x)}{f(x)}
\pen{.5pt}
\dashed\lines{(-2,0),(-2,f(-2))}
\dashed\lines{(0,-2),(-2,f(-2))}
\dashed\lines{(1,0),(1,f(1))}
\dashed\lines{(0,1),(1,f(1))}
\tlabel[tl](2.1,2){$y=x$}
\tlabel[cl](2,-1){$y=2-x^2$}
\tlabel[tr](-2.1,-.1){$-2$}
\tlabel[br](-sqrt 2,.1){$-\sqrt{2}$}
\tlabel[tl](1,-.1){1}
\tlabel[bl](sqrt 2,.1){$\sqrt{2}$}
\tlabel[cl](0.2,-2){$-2$}
\tlabel[cl](0.2,-1){$-1$}
\obd{[cr](-.1,1){1}}
\tlabel[cr](-.1,2.2){2}
\endmfpic
```

3.4 Množiny bodů dané vlastnosti

Množina všech bodů X, z nichž vidíme úsečku AB pod úhlem α.

U tohoto obrázku si všimněme zejména vyznačování úhlů a tvorby čerchované čáry.




```
\mfpicunit=30pt
\mfpic[1]{-4}{4}{-6}{6}
\mfsrc{pair A,B,C,D,S,O,X,Y,Z;
A=(-2,0); B=(2,0);      alpha:=35;
C=3*dir(-alpha)+A;      D=4*dir(-alpha)+A;
S=A+whatever*dir(90-alpha);    S=whatever*up;    O=-S;}
\lines{A,B}     \lines{A,D}     \point{A,B,C,S,O}
\dashed\lines{A,S}   \dashed\lines{B,S}
```

```
\dashpattern{cerchovana}{10pt,4pt,0pt,4pt}
\gendashed{cerchovana}\lines{(0,-5),(0,5)}
\point{.3*dir(45-alpha)+A}
\arc[p]{A,-alpha,90-alpha,.6}
\arc[p]{A,-alpha,0,.7}
\point{(.35*dir 225)}
\arc[p]{.5[A,B],180,270,.7}
\arc[p]{S,270-alpha,270,.7}
\draw[.65white]\arc[p]{S,alpha-90,270-alpha,abs(S-A)}
\coords
\mirror{A}{B}
\draw[.65white]\arc[p]{S,alpha-90,270-alpha,abs(S-A)}
\endcoords
\mfsrc{X=S+abs(S-A)*dir(77);}
\pen{1.3pt}     \polygon{A,B,X}
\pen{.5pt}
%\mfsrc{path a;   %%% vyznačení úhlu lze nakreslit také takto
%a=fullcircle scaled 42pt shifted zconv (X);
%draw subpath (ypart (zconv ((X--A)) intersectiontimes a),
%ypart(zconv((X--B)) intersectiontimes a)) of a;}%
\mfsrc{x=angle(X-A); y=angle(X-B);}
\arc[p]{X,x+180,y+180,.7}
\tlabel[cr](xpart A -.3,ypart A){$A$}
\tlabel[cl](xpart B +.3,ypart B){$B$}
\tlabel[bl](xpart C +.1,ypart C){$C$}
\tlabel[bl](xpart S +.1,ypart S){$S_1$}
\tlabel[cl](xpart O +.1,ypart O){$S_2$}
\tlabel[bl](0.1,0.1){$M$}
\tlabel[bc](xpart X,ypart X +.1){$X$}
\tlabel[br](xpart .5[A,S] -.1,ypart .5[A,S]){$r$}
\tlabel[bl](xpart .5[B,S] +.1,ypart .5[A,S]){$r$}
\mfsrc{Y=S+abs(S-A)*dir(60);}
\tlabel[bl](xpart Y +.1,ypart Y){$k_1$}
\tlabel[tl](xpart Y +.1,-ypart Y){$k_2$}
\mfsrc{pair f,g,h;f=.9*dir((x+y)/2+180)+X;
                  g=.9*dir(-alpha/2)+A;
                  h=.9*dir(270-(alpha/2))+S;}
\tlabel[cc](xpart f,ypart f){$\alpha$}
\tlabel[cc](xpart g,ypart g){$\alpha$}
\tlabel[cc](xpart h,ypart h){$\alpha$}
\endmfpic
```

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 TEXu 2/94*. Brno. http://bulletin.cstug.cz/pdf/bul942.pdf

[6] Olšák, P. *TEXbook 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. *LATEX pro začátečníky*. Brno: Konvoj, 2. vydání, 1999. ISBN 80-85615-74-6

Zajímavé odkazy týkající se tématu práce:

- 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 příkazů, parametrů a proměnných pro část METAPOST

Rejstřík příkazů a voleb pro část mfpic