Pracovné poznámky

Podporné aplikácie pre LaTex

Circuit-Macros

Inštalácia a príklady

Verzia 0.3

pf

7. januára 2021

Obsah

Kapitola 1

Circuit-Macros

Knižnica Circuit-Macros (CM) je je rozšírenim programovacieho jazyka *dpic* (určného pre kreslenie diagramov a grafov, staršia verzia je označovaná ako *gpic*) pomocou súboru makier pre makroprocesor *m4*. Pomocou tohoto rozšírenia je možné vytvárať elektrické zapojenia a schémy, zároveň je možné v nich používať aj grafické prvky jazyka *dpic*.
Makrá expandujú značky elektronických prvkov do množiny príkazov jazyka *pic*, výsledný program je pomocou kompilátora preložený do grafických príkazov vektorovej grafiky (*.ps*, *.svg* . . .) alebo do makier *pstricks* pre priame vloženie do dokumentov LaTeX-u.
Kreslenie zapojení je možné vytváraním textového popisu obvodu

- pomocou integrovaného editora a prehliadača vytvorného obvodu PyCirkuit
- pomocou ľubovolného textového editora a následne preložením pomocou príkazov z konzoly do zvoleného vektorového formátu a prípade potreby jeho konvertovaním do bitového formátu
- niektorým z predchádzajúcich spôsobov vytvorením súboru s *pstricks* makrami, ktoré sa vložia do LaTex-vého zdrojového textu dokumentu

Knižnica CM poskytuje aj možnosť programového generovanie schém a zapojení napr. ako vizualizácia výsledoku optimalizačných programov, náhradných schém a pod.

1.1 Inštalácia

Knižnica *circuit-macros* je súčasťou štandardných repozitárov

```
sudo apt-get install circuit-macros
```

Inštalácia a popis programu *PyCirkuit* je uvedená v Ďaľšej kapitole.

Závislosti a podporné programy

Podporné programy a utility

```
m4          - štandardná súčasť distribúcií Linuxu
dpic        - kompilátor jazyka pic - súčasť distribúcií
latex       - s podporou pre pstricks a podpornými programami (dvips)
imagemagick - pre konverziu grafických formátov
```

Poznámka k *imagemagick*

Pre konverziu vektorových do bitových formátov vyžaduje program *convert* z balíku *imagemagick* povolenie pre konverziu *ps, eps a pdf* súborov, štandardne je táto konverzia zakázaná. Povolenie je možné nastaviť v súbore *policy.xml*:

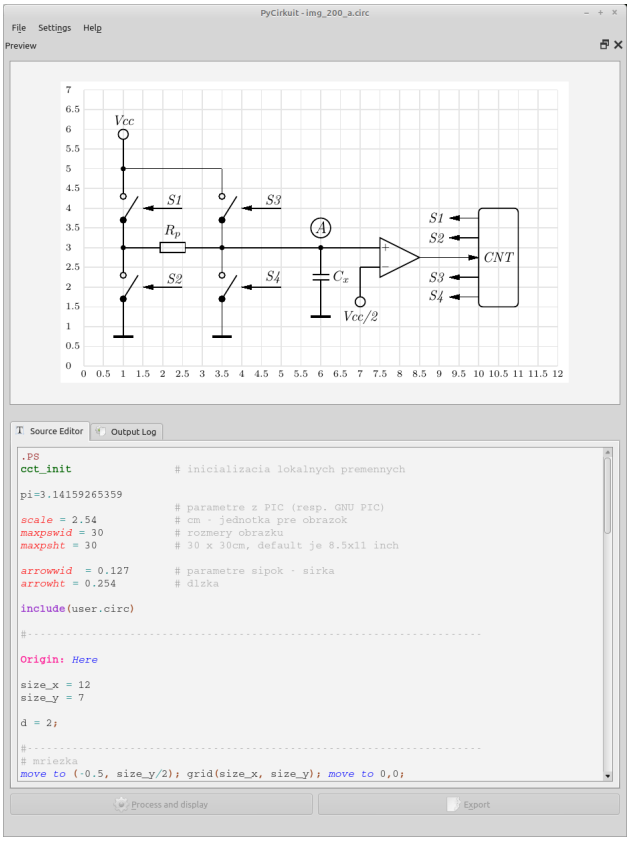
remove this whole following section from /etc/ImageMagick-6/policy.xml:

```
<!-- disable ghostscript format types -->
<policy domain="coder" rights="none" pattern="PS" />
<policy domain="coder" rights="none" pattern="EPS" />
<policy domain="coder" rights="none" pattern="PDF" />
```

Kapitola 2

PyCirkuit

PyCirkuit je aplikácia obsahujúca jednoduchý editor zdrojových textov pre CM s farebným zvýrazňovaním syntaxe, kompilátorom a prehliadač vytvorených schém a zapojení. Program umožňuje exportovať vytvorené zapojenia do štandardných grafických formátov ako aj do makier *pctricks*.



Obr. 2.1: PyCirkuit

Domáca stránka programu je na

`https://github.com/orestesmas/pycirkuit`

PyCirkuit je do Pythonu prepísaná verzia programu Cirkuit, jeho originálna zdrojová stránka je na

`https://wwwu.uni-klu.ac.at/magostin/cirkuit.html`

2.1 Inštalácia

Program je dostupný v štandardných repozitároch. Pri inštalácii si automaticky doinštaluje aj knižnicu *circuit-macros* a príslušné podporné programu (*dpic*). *m4* makroprocesor je štandardnou súčasťou distríbúcie Linux-u.

```
sudo apt-get install pycirkuit
```

Overené na Mint 20.x, na starších verziách generuje chyby pri kompilacii obrázku.

Kapitola 3

Použitie *circuit-macros* z konzoly

Proces vytvárania zapojenia, jeho komplácie a konverzie do výsledného grafického formátu môžeme urobiť aj ručne priamo z konzoly systému. V nasledujúcom texte sú použité označenia

```
PATH_TO_CM_MACROS - cesta k inštalácii circuit-macros
SOURCE_FILE       - zdrojový textový súbor s vytvoreným zapojením
PIC_FILE          - vystupny subor predpropcesora m4 s prikazmi gpic
PSTRICK_FILE      - kompilovanu subor s prikazmi pre pstricks
LATEX_FILE        - finalny LaTeX súbor
```

3.1 Preklad a kompilácia s využitím LaTex-u

Najprv uvedieme komplikovanejší postup s využitím renderovania textov v LaTex-e a s využitím makier PSTricks, predpokladáme, že LaTex s príslušnými modulmi máte už nainštalovaný. V prvom kroku preložíme naše zapojenie pomocou makroprocesora do príkazov jazyka PIC

```
m4 -I PATH_TO_CM_MACROS pstricks.m4 SOURCE_FILE > PIC_FILE
```

v príkaze zadefinujeme cestu k adresáru, kde sme rozbalili makrá z archívu, náš zdrojový súbor so zapojením a zvolíme meno súboru, do ktorého sa uložia príkazy v PIC. *pstricks.m4* obsahuje sadu pomocných makier pre finálny výstup.
V druhom kroku súbor v jazyku PIC do sady makier *pstricks* preložíme pomocou

```
dpic -p PIC_FILE > PSTRICK_FILE
```

Výsledkom prekladu je zdrojový text pre LaTexu-u s príkazmi pre *pstricks*, ktorý má štandardný tvar

```
\begin{pspicture}  ...  \end{pspicture}
```

Skompilovaný súbor už môžeme vložiť do dokumentu v LaTex-u a ďalej s ním pracovať. Pretože pri kreslení schém potrebuje priebžne sledovať výsledok práce, pre renderovanie obrázku zapojenia do postscriptu môžeme použiť krátky dokument (LATEX_FILE)

```
\documentclass{article}
\usepackage{times,pstricks,pst-eps,pst-grad}
\usepackage{graphicx}
\begin{document}
\begin{TeXtoEPS}

\input PSTRICK_FILE        <- menu skompilovaneho suboru bez pripony

\end{TeXtoEPS}\end{document}
```

ktorý preložíme

```
latex LATEX_FILE      <- bez pripony .tex
dvips -E LATEX_FILE   <- bez pripony .aux
```

Výsledkom prekladu je obrázok vo formáte postscript. Aj keď uvedený postup vyzerá komplikovane, jednoduché skripty v pythone alebo shell-e problém vyriešia.

3.2 Preklad a kompilácia bez využitia LaTex-u

Pri jednoduchšom spôsobe grafický súbor vygeneruje priamo kompilátor *dpic*, samozrejme ale bez renderovania textov LaTex-om.

```
m4 -I PATH_TO_CM_MACROS postscript.m4 SOURCE_FILE | dpic -r > PS_FILE
```

Kapitola 4

circuit-macros v jupyter-notebooku

Aby bolo možné písať manuály k *circuit-macros* s demonštračnými obrázkami v notebooku, bola vyvorená podporná knižnica *cm_utils*, obsahujúca jednoduchý template pre dokumenty a pomocné funkcie pre riadenie kompilácie a zobrazenie obrázkov priamo v notebooku. Pre tvorbu schém je vytvorené template s definíciami pre začiatok a koniec súboru so zapojením tak, aby bolo možné v notebooku editovať len vlastné zapojenie.

4.1 Obsah pomocnej knižnice *cm_utils*

Template pozostáva z textového reťazca pre záhlavie (*cm_start*)a koniec súboru (*cm_end*). Záhlavie obsahuje inicializáciu, nastavenie rozmerov obrázkov a zobrazenie pomocnej mriežky pre kreslenie. Funkcia *cm_show()* vytvorí súbor so zapojením, skompiluje zapojenie a vyvolá zobrazenie obrázku v notebooku. Pri klasickom spôsobe zobrazenia pomocou hrml-tagov totiž nedôjde pri zmene obrázku k jeho aktualizácii, pretože sa zobrazuje verzia uložená v cache prehliadača.

```
cm_start = '''              # zahlavie suboru
.PS
scale = 2.54                # cm - jednotka pre obrazok
maxpswid = 30               # rozmery obrazku
maxpsht = 30                # 30 x 30cm, default je 8.5x11 inch
cct_init                    # inicializacia lokalnych premennych

arrowwid  = 0.127           # parametre sipok - sirka
arrowht = 0.254             # dlzka

include(./img/user.ckt)     # externa kniznica komponentov
                            # *** upravit podla aktualnej cesty ***

size_x = 10
size_y = 6
d = 2;                      # 1cm zakladna dlzka, d=2cm

# mriezka
move to (-0.5, size_y/2); grid(size_x, size_y); move to 0,0;
'''

cm_end = '''                # template pre koniec suboru
.PE
'''

def cm_show(file_name, cm_data='', width=400):
    '''
    Funkcie pre preklad a zobrazenie obrazku v notebooku
    '''
    fp = open('./img/' + file_name + '.ckt', 'w');
    fp.write(cm_start + cm_data + cm_end);
    fp.close()
```

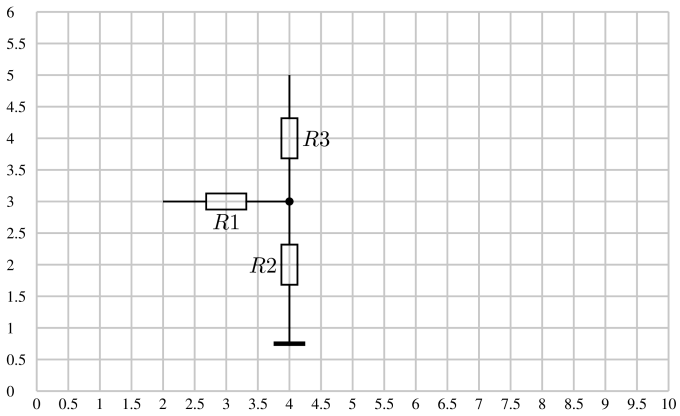
```
    cm2ps('./img/' + file_name + '.ckt' );
    show_png('./img/' + file_name + '.png', width)
```

4.2 Použitie v notebooku

Príklad vytvorenia popisu zapojenia v textovom reťazci, jeho kompilácia a zobrazenie. Pri kompilácii sa vytvárajú v adresári *./img/* obrázky vo formáte *.eps* a *.png*.

```
from img.cm_utils import *

data='''                                # vlastne zapojenie ako textovy retazec
move to (2,3)                           # bez zahlavia a konca suboru
R1: resistor(d,E); rlabel(,R1,);
dot;
{
    up_;
    R3: resistor(d,E); rlabel(,R3,);
}
down_;
R2: resistor(d,E); rlabel(,R2,);
gnd;
'''

cm_show('obr-04', data, 400)            # kompilacia a zobrazenie
```

Zobrazenie vygenerovaného obrázku vo formáte *.png*.



Obr. 4.1: Obrázok z príkladu

Kapitola 5

Syntax

5.1 Komentáre

Komentáre začínajú znakom # do konca riadku ukončeného znakom
n. Blokové komentáre nie sú definované, je ale možné použiť viacriadkové komentáre s riadkami ukončenými

.

```
# toto je jednoriadkovy komentar
# toto je dvojriadkovy komentar        \\
  a jeho pokracovanie na dalsom riadku
```

White-space sú ignorované

```
name( x,
y, z )
```

je ekvivalent

```
name(x,y,z)
```

5.2 Hodnoty

Numerické hodnoty môžu obsahovať desatinnú bodku alebo môžu byť vo vedeckom formáte. Všetky numerické hodnoty sú interne uchovávané vo formáte floating-point. Hodnoty súradníc sú zapísané usporiadanou dvojicou *(x,y)*, každá súradnica impicitne obsahuje atribúty *x* a *y*.

5.3 Premenné

Meno premennej skalárnej musí začínať písmenom nasledovaným ľubovolným počtom alfanumerických znakov. Premenné sú globálne a majú platnosť v celom kóde.

```
d=2;
pi=3.14159265359;
q=2*pi*8;
```

Súradnice sú reprezentované ako dvojice (x,y) a nemôžu byť použité ako hodnoty premennej.
Textové premenné sú definované v obyčajných úvodzovkách

```
str = "Toto je text";
```

Po inicializácii skriptu vyvolaním *cct_init* sú vytvorené preddefinované premenné, ktorými je možné nastavovať parametre kreslenia, napr.

```
Here      - posledná poloha (read only)

scale     - definuje jednotky, štandardne je 1 inch
arrowwid  - parametre sipok - sirka
arrowht   - parametre sipok - dlzka
...
```

Konflikt mien

- Niektoré makrá definujú premenné, ktoré môžu byť príčinou konfliktov. Napr makro *setrgb()* vytvára premenné $r\_$, $g\_$, $b\_$, kde prvá vytvorí konflikt, ak potrebujeme označiť rezistor pomocou syntaxe v LaTex-u napr. *r_1*. V takomto prípade je potrebné v reťazci pre LaTeX použiť formálne prerušenie reťazca *r _1*.
- Nie je možné v zobrazovanom texte použiť mená makra, pretože pri substitúcii dôjde k nahradeniu textu a následnému konfliktu.

5.4 Referencie

Každá entita v CM môže byť označená referenciou, prostredníctvom ktorej je možné odkazovať sa na jej atribúty (ak sú definované). Meno referencie musí začínať veľkým písmenom nasledovaným ľubovolným počtom alfanumerických znakov.

```
R1: resistor();
```

Referenciou je možné označiť aj súradnice.

```
Stred: (5,6);
```

Pomocou referencií je možné pristupovať k individuálnym atribútom komponentov, napr:

```
Stred.x  - má hodnotu 5
Stred.y  - má hodnotu 6
```

5.5 Skript

Skript v jazyku *pic* začína postupnosťou *.PS* nasledovannou nepovinnými parametrami s rozmerom obrázka a končí *.PE*.

```
.PS [width [height]]

    instrukcia;
    ...

.PE
```

Text za koncom skriptu je ignorovaný. Ukončenie skriptu môžeme prakticky využiť pri hľadaní chýb v skripte, kedy pomocou *.PE* vyradíme zbytok skriptu zo zobrazovania.

5.6 Inštrukcie a príkazy

Inštrukcia (statement ke jeden alebo viacej príkazov) končiacich znakom bodkočiarky *;* alebo znakom konca riadku. Je vhodné implicitne používať znak konca riadku vždy, pri prípadnom dopĺňaní skriptu sa obmedzí vznik chýb.

```
resistor() ;    # inštrukcie končí znakom ;
capacitor()     # inštrukcia končí \n
```

5.7 Bloky

Premenné uzatvorené v bloku $\{\dots\}$ majú lokálnu platnosť, referencie na polohy majú platnosť globálnu. Kód v bloku sa vzťahuje k poslednej aktuálnej polohe a nemení ju, je preto používaný na kreslenie vetiev obvodov vzľadom k referenčnej polohe.

```
d = 2;
{
    d = 0.4;
    Q: (1,1);          # globálná definícia polohy
    ...
}
line from Q right_ d   # d má hodnotu 2
```

5.8 Riadenie toku

Cyklus

Jednoduchý cyklus s premennou x má tvar

```
for x = 1 to 10 by 2 do { line from (0,0) to (5,x); }
```

kde v zložených zátvorkách je telo cyklu, toto má vlastnosti bloku s relatívnymi súradnicami vztiahnutými k začiatku cyklu.

Vetvenie

Základný formát vetvenia toku programu má tvar

```
if expression then { if-true } else { if-false }
```

Kapitola 6

Aritmetika

V jazyku dpic sú definované základné matematické operácie pre skalárne a vektorové (2D súradnice) premenné.

6.1 Matematické operácie

```
+
-
*
/
%
hat
!=
==
<
>
>=
<=
||
\&\&
```

Unárne operátory

`+=, -=, *=, /=, %=,`

6.2 Vektorová aritmetika

Bod má definovanú polohu dvojicou súradníc (x, y).

```
P1: (1,1)             # atributy P1.x, P1.y
P2: (1,2) + (3,4) - (0.5, 0.5)
P3: (1,1) + P1
P4: P1 + P2
P5: (P3, P4)          # ekvivalent (P3.x, P4.y)
P6: P1*k              # k je skalarna hodnota
```

Pri použití relačných operátorov na súradnice treba príslušnú operáciu realizovať po zložkách

```
P1 > P2               # chyba
P1.x > P2.x           # ok
```

6.3 Zabudované funkcie

V jazyku dpic sú definované základné matematické funkcie pre skalárne premenné.

```
sin(<expr>)
cos(<expr>)
log(<expr>)
exp(<expr>)
sqrt(<expr>)
max(<expr>, <expr>...)
atan2(<expr>, <expr>)
min(<expr>, <expr>...)
int(<expr>)
rand(<expr>)
abs(<expr>)
acos(<expr>)
asin(<expr>)
expe(<expr>)
floor(<expr>)
loge(<expr>)
sign(<expr>)
tan(<expr>)
pmod(<expr>)
```

Kapitola 7

Práca s komponentami

Použitie základných grafických komponentov..

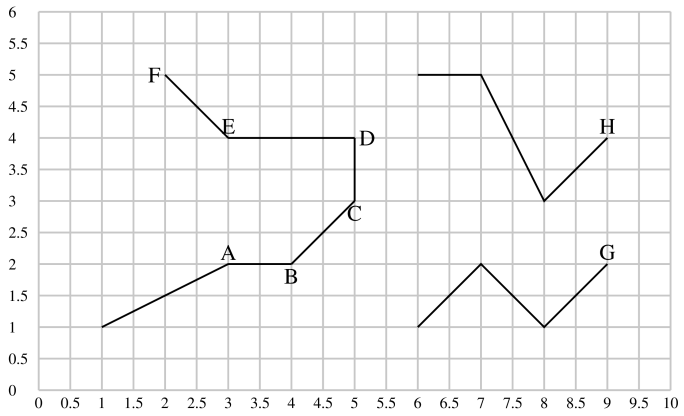
7.1 Čiary, šípky a krivky

V template je vytvorená mriežka s krokom 1x1cm, v súradnicovej sústave sa zadáva poloha dvojicou (x,y). Posledná poloha je označená ako *Here*. Suradnice môžu byť zadané absolútne, relatívne voči poslednej polohe alebo smerom.

```
from img.cm_utils import *

L1='''
line from (1,1) to (3,2); {"A" above};  # A. absolutne polohy bodov, nastavuje
                                        #    poziciu Here na konc. bod
line from Here to (4,2);  {"B" below};  # B. ciara od aktualnej pozicie
line to (5,3);            {"C" below};  # C. to iste od posledneho bodu
line to Here + (0,1);     {"D" ljust};  # D. relativne od poslednej pozicie
line left_ 2              {"E" above};  # E. relativne zadanim smeru v jednej
    osi
line left_ 1 up_ 1;       {"F" rjust};  # F. relativne v dvoch osiach

                                        # G. zadanim postupnosti bodov
line from (6,1) to (7,2) to (8,1) to (9,2); {"G" above};

                                        # H. postupnostou relativnych krokov
line from (6,5) right_ 1 then right_ 1 down_ 2 then right_ 1 up_ 1; {"H" above};

'''

cm_show('obr-100', L1, 500)
```

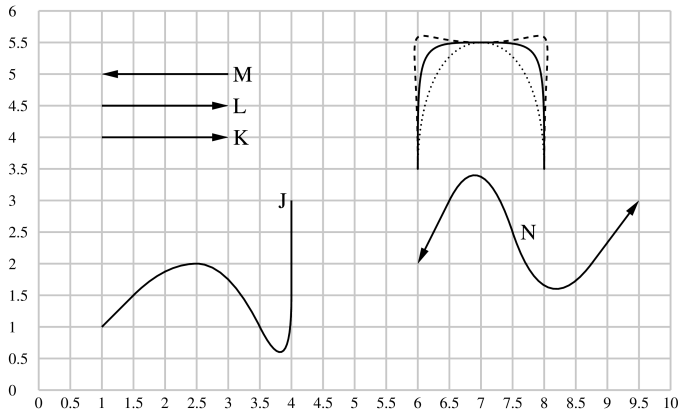
Krivky môžeme kresliť rôznymi spôsobmi, pre krivky definované ako spline môžeme nastaviť parametrom tvar krivky (tension parameter). Čiary aj krivky mpžeme modifikovať parametrami *dashed* a *dotted*, za ktorými môže nasledovať numerická hodnota udávajúca hustotu čiarok alebo bodiek (závisí od zvolenej mierky obrázku).

```
L2='''
                                        # J. spline krivka, suradnice
                                        #    rovnake ako pri ciare
spline from (1,1) right_ 1 up_ 1 then right_ 1 down _1 then right_ 1 down_ 2
    then up_ 3; {"J" rjust};

arrow from (1,4) right_ 2;     {"K" ljust};  # K. sipka menom
line -> from (1,4.5) right_ 2; {"L" ljust};  # L. sipka smerom doprava
line <- from (1, 5) right_ 2;  {"M" ljust};  # M. sipka smerom dolava

                                        # N. obojstranna sipka, oznacenie
                                        #     v strede
S1: spline <-> from (6,2) to (7,4) to (8,1) to (9.5,3); {"N" at S1.c};
```



Obr. 7.1: Skript L1

```
spline 1.4 from (6, 3.5) up_ 2 then right_ 2 then down_ 2 dashed .08;
spline 1.0 from (6, 3.5) up_ 2 then right_ 2 then down_ 2;
spline 0.6 from (6, 3.5) up_ 2 then right_ 2 then down_ 2 dotted .05;
'''

cm_show('obr-200', L2, 500)
```

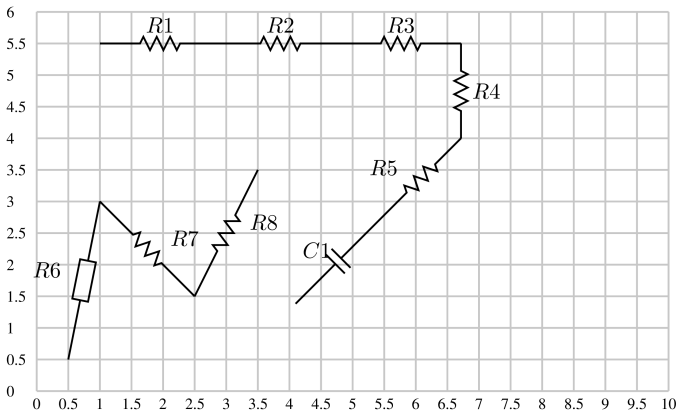


Obr. 7.2: Skript L2

7.2 Ukladanie komponentov

Pre uložením komponentu treba definovať smer, default je *right_*. V CM sa pre určenie smeru používajú označenia smeru s podtržítkom, ktoré nastavujú hodnoty globálnych premenných (napr. *Here*). Rovnaké mená, ae bez podtržítka používa *dpic*.
Makrá *rlabel* a *llabel* slúžia na popis dvojpólov vo formáte (začiatok, stred, koniec), poloha popisu je v smere ukladania komponentu. Text môže byť v matematickom formáte LaTeX-u.

```
L3='''
move to (1, 5.5);

right_;                      # vseobecne nastavenie smeru
resistor();                   llabel(,R1,);
resistor();                   llabel(,R2,);
resistor();                   llabel(,R3,);

# nastavenie smeru a dlzkou, nastavuje aj globalny smer
resistor(down_ 1.5); llabel(,R4,);

# nastavenie specifickeho smeru pre dvojpoly
resistor(down_ left_ );      rlabel(,,R5);
capacitor();                 rlabel(,,C1);

# zadanie oboma koncovymi bodmi
resistor(from (0.5, 0.5) to (1,3), E); rlabel(,,R6);

resistor( to (2.5, 1.5) );  rlabel(R7,,);       # len koncovym bodom od poslednej
                                                # polohy, absolutne

resistor( to Here+(1,2));   rlabel(R8,,);       # relativnou velkostou
                                                # k poslednej polohe Here
'''

cm_show('obr-300', L3, 500)
```



Obr. 7.3: Skript L3

7.3 Atribúty komponentov

Pozícia v CM môže byť definovaná aj voči atribútom komponentu. Názvy atribútov sú definované v makre príslušného komponentu, k atribútom sa pristupoje cez ".". Pre čiaru sú definované štandardné atribúty

```
start    alebo s
center          c
end             e
```

Atribút obsahuje dvojici súradníc (x,y), ku ktorým sa pristupuje identicky, pre čiaru L

```
L.center
```

je ekvivalentom

```
(L.center.x, L.center.y)  alebo (L.c.x, L.c.y)
```

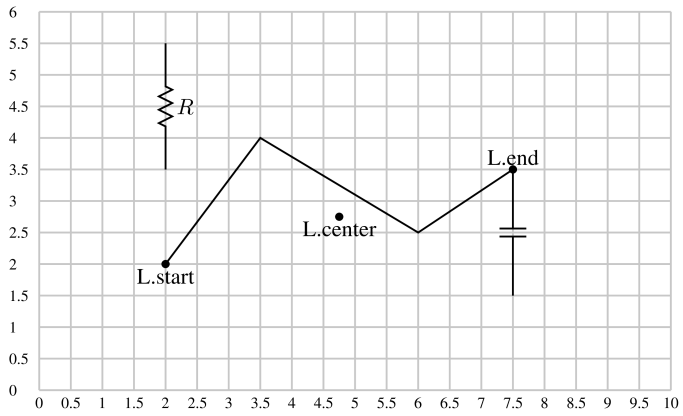
Pri pozičnom použití atribútov sa automaticky vyberá príslušná zložka

```
(L.s, L.e)
```

je ekvivalentom

```
(L.s.x, L.e.y)
```

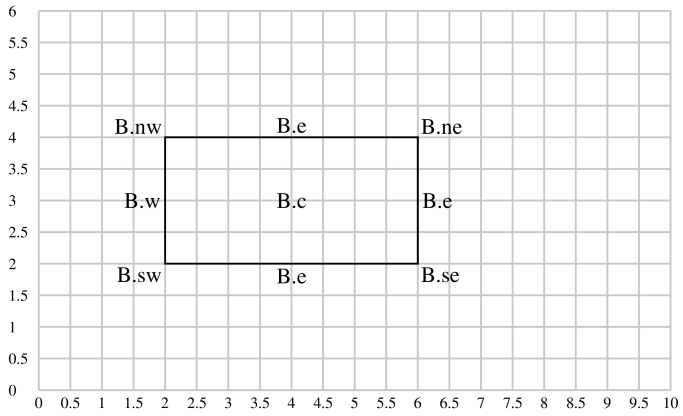
```
L4='''
move to (2, 2);

L: line right_ 1.5 up_ 2 then right_ 2.5 down_ 1.5 then right_ 1.5 up_ 1

move to L.start;  dot; "L.start"  below;        # rovnako ako L.s
move to L.center; dot; "L.center" below;
move to L.end;    dot; "L.end"    above;

capacitor(from L.e down_ 2)                     # umiestnenie relativne k ciare
resistor( from (L.s, L.e) up_ 2 ); rlabel(,R,);
 '''

cm_show('obr-400', L4, 500)
```



Obr. 7.4: Skript L4

Štandardné atribúty pre box - podľa svetových strán.

```
L5 = '''
move to (2, 3);

B: box wid 4 ht 2

move to B.west;   "B.w"  rjust;    # rovnako ako B.w
move to B.c;      "B.c";
move to B.e;      "B.e"  ljust;
move to B.n;      "B.e"  above;
move to B.s;      "B.e"  below;

move to B.ne;     "B.ne"  above ljust;
move to B.nw;     "B.nw"  above rjust;

move to B.se;     "B.se"  below ljust;
move to B.sw;     "B.sw"  below rjust;
 '''

cm_show('obr-500', L5, 500)
```

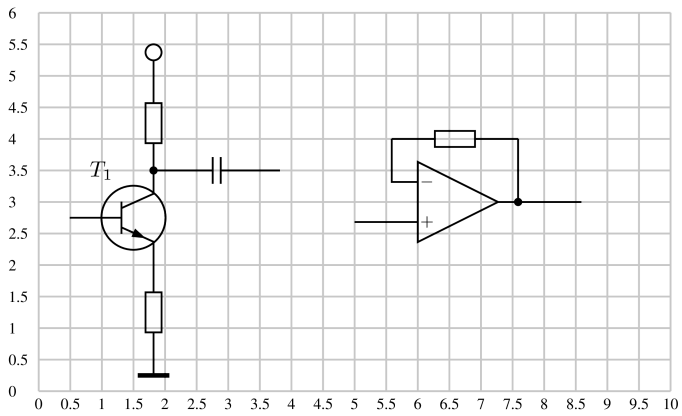


Obr. 7.5: Skript L5

Komplikovanejšie komponenty majú okrem štandardných atribútov definované aj vlastné atribúty, napr. pri bipolárnom tranzistore B,C.E a pod. Zoznam atribútov pre daný komponent je dostupný v dokumentácii ku komponentu.

```
# pouzitie specifickych atributov komponentu
L6 = '''
move to (1.5, 2);
up_;
T: bi_tr(1.5, L, N,E); {"$T_1$" at T.nw}
resistor(from T.E down_ 1.5, E); gnd;
resistor(from T.C up_ 1.5, E); power;
line from T.B left 0.5;

move to T.C;
dot;
capacitor(right_ 2)

#---------------------------------------

move to (6,3);
OP: opamp()
line from OP.In2 left_ 1
move to OP.Out;
dot;
```

```
{ line right_ 1 }
line up_ 1;
resistor(left_ 2, E);
line to (Here.x, OP.In1.y) to OP.In1
'''

cm_show('obr-600', L6, 500)
```



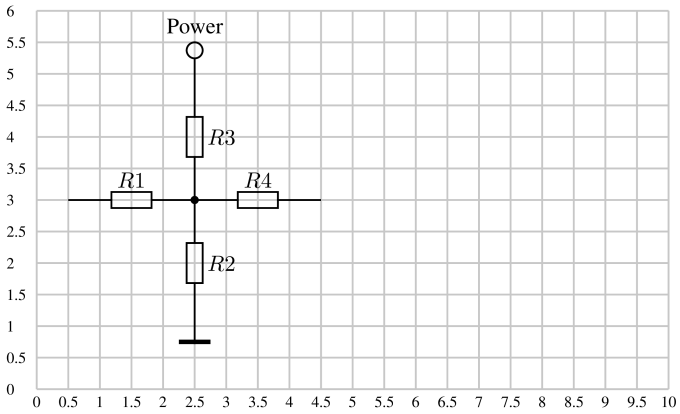
Obr. 7.6: Skript L6

Kapitola 8

Bloky

Uzatvorenie časti kódu do blokov umožňuje vytváranie vetvy obvodu alebu umiesnenie iných komponentov relatívne k poslednej súradnici bez toho, aby bola táto zmenená. Bloky je možné vnárať.

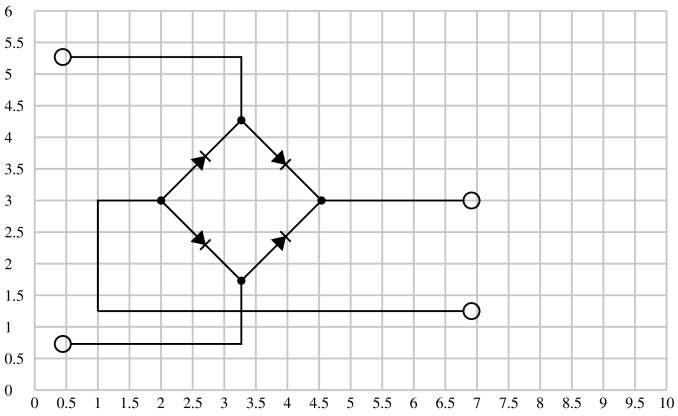
```
from img.cm_utils import *


S1 = '''
d=2;
move to (0.5, 3);
resistor(right_ d,E);         llabel(,R1,);
dot;
{
    resistor(down_ d, E);     llabel(,R2,);
    gnd;
}
{
    resistor(up_ d,E);        rlabel(,R3,);
    P: power();     {"Power" at P.n above};  # vnoreny blok

}
resistor(right_ d,E);         llabel(,R4,);
'''

cm_show('obr-317-1', S1, 500)
```



Obr. 8.1: Skript S1

Referencie vytvorené vo vnútri bloku sú globálne a je možno sa na ne odkazovať v nasledujúcom kóde ako aj v iných blokoch.

```
S2 = '''
d=2;
move to (2, 3);
DT1: dot;                    # pociatocny bod
{
    diode(right_ up_);       # horna vetva diod
    DT2: dot;                # vytvorena referencia
    diode(right_ down_);
}

{
    diode(right_ down_);     # dolna vetva diod
    DT3: dot;
    diode(right_ up_);
    DT4: dot;
}

L1: line from DT2  up_ d/2 then left_ d;   tconn(,0);
L2: line from DT3  down_ d/2 then left_ d; tconn(,0);
L3: line from DT4 right_ d; tconn(d/4, 0);

L4: line from DT1 left_ d/2 then down_ 7*d/8;
    line to (L3.e.x, Here.y); tconn(right_ d/4,0);

'''

cm_show('obr-317-2', S2, 500)
```



Obr. 8.2: Skript S2

Kapitola 9

Text

Základné formátovanie textu je možné priamo v CM, komplexnejšie formátovanie je možno z využitím renderovania textu v LaTeX-u.

9.1 Pozícia textu

Text je ohraničený obyčajnými uvodzovkami nasledovanými určením polohu textu a relatívneho umiestnenia voči tejto polohe.

```
"Test" location position;
```

Uminestnenie (position) môže byť

```
ljust
rjust
above
belov
```

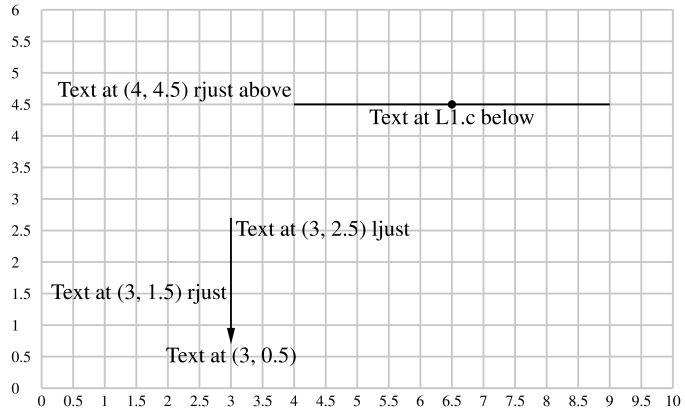
Text modifikuje globálnu premennú *Here*

Použitie textu s atribútami objektov

```
from img.cm_utils import *

sch=r'''
line <- from (3, 0.7) up_ 2;
"Text at (3, 0.5)" at (3, 0.5);
"Text at (3, 1.5) rjust"  at (3, 1.5)  rjust;
"Text at (3, 2.5) ljust"  at (3, 2.5)  ljust;

L1: line from (4, 4.5) to (9.0, 4.5)
move to L1.c; dot;
"Text at (4, 4.5) rjust above"  at (4, 4.5)  rjust above;
"Text at L1.c below"  at L1.c below;
'''

cm_show('obr-319-1', sch, 600)
```

9.2 Matematické výrazy

Pre zobrazenie matematické výrazy sa používa štandardné formátovanie LaTeX-u. Text sa korektne zobrazí len pri renderovaní pomocou LaTeX-
Doplenia pre notebook

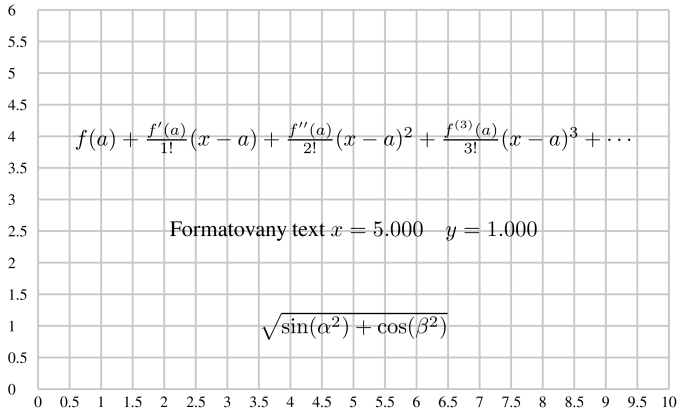


Obr. 9.1: Skript

- pretože v matematickom texte sa vyskytujú spätné lomítka, je potrebné textový reťazec s matematickými výrazmi deklarovať ako **raw** text.
- nepoužívať pre tretiu deriváciu znak 3x apostrof, pretože to je znak ukončenia dlhého reťazca

Funkcia *sprintf* akceptuje len formatovacie znaky **e,f,g**, nie je možné tlačiť string.

```
sch=r'''
T: "$\sqrt{\sin(\alpha^2) + \cos(\beta^2)}$" at (5, 1);

"$f(a)+\frac {f'(a)}{1!} (x-a) +
       \frac{f''(a)}{2!} (x-a)^2 +
       \frac{f^{(3)}(a)}{3!}(x-a)^3 + \cdots$" at (5,4);

sprintf("Formatovany text $x=%2.3f$ \,\,\,  $y=%2.3f $", T.x, T.y) at (5, 2.5);
'''

cm_show('obr-319-2', sch, 600)
```



Obr. 9.2: Skript

9.3 Formátovanie textu

Jaxyk *dpic* poskytuje len obmedzené možnosti formátovania textu. Ak je pre renderovanie pokužitý LaTeX, je možné používať pre úpravu textu jeho prostredia makrá

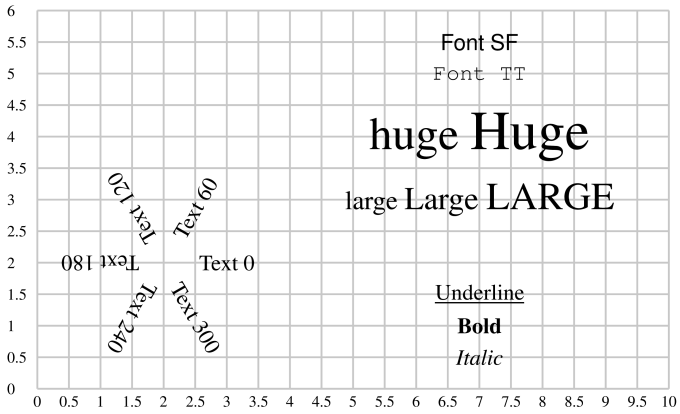
```
\textit{text}
\textbf{text}
\underline{text}
```

Pre rotáciu textu je potrebné použiť makro

```
\rotatebox{angle}{text}
```

Poloha otočeného textu je vždy v jeho strede, pridávanie medzier na začiatok alebo koniec textu neposunie stred textu.

```
sch=r'''
"\textit{Italic}" at (7, 0.5);
"\textbf{Bold}" at (7, 1);
"\underline{Underline}" at (7, 1.5);

for i=0 to 300 by 60 do {
    move to ( 2 + 1*cos(i/180*pi), 2 + 1*sin(i/180*pi) );
    sprintf("\rotatebox{%g}{Text %g}", i,i );
}

"\large large \Large Large \LARGE LARGE" at (7,3);
"\huge huge \Huge Huge" at (7,4);

"\texttt{Font TT}" at (7,5);
"\textsf{Font SF}" at (7,5.5);
'''

cm_show('obr-319-3', sch, 600)
```



Obr. 9.3: Skript S2

Kapitola 10

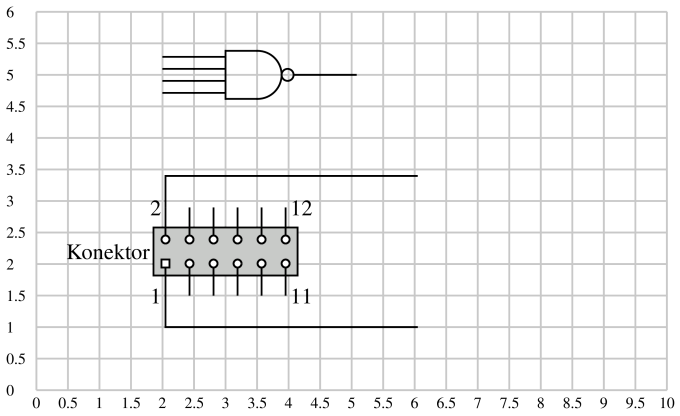
Parametrické komponenty

V CM je možné vlastnosti niektorých komponentov definovať pomocou parametrov, zvyčajne u komponentov, ktoré obsahujú opakujúce sa časti, napríklad konektory, vstupy logických hradiel a pod.

```
from img.cm_utils import *

sch='''
log_init;          # inicializacia makier pre logicke obvody
d=2;

move to (3,1.5);
up_;
H1: Header(2, 6,,,fill_(0.9));
"Konektor" at H1.w rjust;
"1" at H1.P1 rjust;
"2" at H1.P2 rjust;
"11" at H1.P11 ljust;
"12" at H1.P12 ljust;

line from H1.P1 down_ d/4 then right_ 2*d;
line from H1.P2 up_ d/4 then right_ 2*d;


move to (3, 5);
right_; G1: NAND_gate(4);

line from G1.In1 left_ d/2;
line from G1.In2 left_ d/2;
line from G1.In3 left_ d/2;
line from G1.In4 left_ d/2;
line from G1.Out right_ d/2;
'''

cm_show('obr-320-1', sch, 500)
```

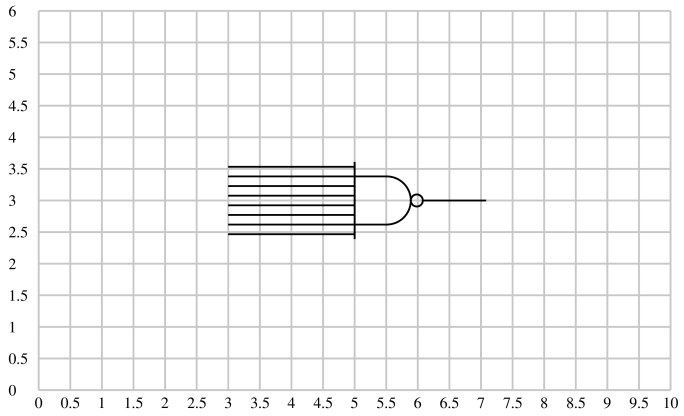
Pri parametrických komponentoch môžeme použiť dynamické vytváranie mien atribútov v textovom reťazci obsahujúcom príkaz. Reťazec môžeme vytvoriť parametricky pomocou funkcie *sprintf()*. Textový reťazec vykonáme pomocou príkazu *exec*.

```
sch=r'''
log_init;
d=2;

move to (5,3);
right_;
G2: NAND_gate(8);

for i=1 to 8 do {
```



Obr. 10.1: Skript

```
    exec sprintf("line from G2.In%g left_ d",i);
}

line from G2.Out right_ d/2;
'''

cm_show('obr-320-2', sch, 500)
```



Obr. 10.2: Skript

Kapitola 11

Vytváranie uživateľských makier

Makrá sa definujú podľa syntaxe makroprocesora *m4*. Všeobecný tvar makra je

```
define (name, [expansion])
```

Reťazec *name* je nahradený reťazcom *expansion*. Príklad

```
define(‘foo’, ‘Hello world.’)
```

po spracovaní dostaneme

```
foo
Hello world.
```

Makro môže mať argumenty, tieto sú označované ako $1, $2 . . . , špaciálny význam má $0, ktorá obsahuje meno makra. Príklad makra, ktoré vymení poradie argumentov

```
define(‘exch’, ‘$2, $1’)
```

po spracovaní dostaneme

```
exch(‘arg1’, ‘arg2’)
arg2, arg1
```

Na základe hodnôt parametron môžeme makrá parametrizovať. Nižšie je uvedené makro pre zobrazenie spínača s parametrami dĺžka spínača a stav (ON, OFF) spínača. Aby bol komponent presne umiestnený v mriežke bez ohľadu na jeho aktuálne grafické zobrazenie, je vhodné ho umiestniť do neviditeľného boxu s fixnými rozmermi. Pre vonkajší box potom platia štandardné atribúty.

Poznámka

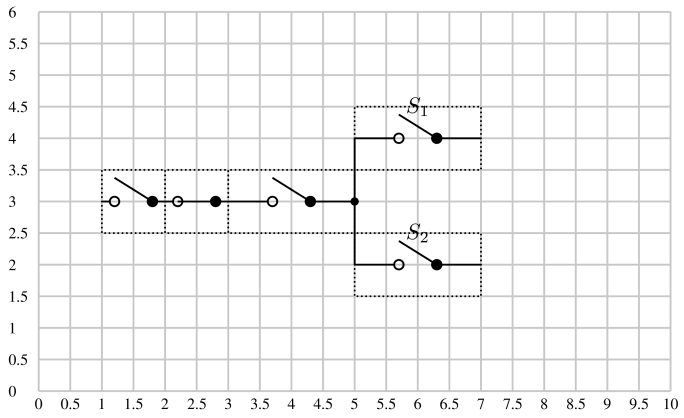
Substitučné reťazce v makrách *m4* začínajú znakom spätného apostrofu chr(96) a končia znakom apostrofu chr(39).

Príklad

Makro definúce spínač v stave ON a OFF s definovateľnou dĺžkou prívodov. Je zobrazený vonkajší box.

```
from img.cm_utils import *

sch=r'''

define(‘swh’,‘[

    B: box ht 1 wid $1 dotted 0.04 #invis;            # neviditelny  box
    rr = 0.15;
    p = 1.5;

    C1: circle diameter rr at  B.c + (rr/2 - p/4, 0)
    C2: circle diameter rr at  B.c + (-rr/2 + p/4, 0) fill 0;
```

```
    line from C1.w to B.w
    line from C2.e to B.e
    ifinstr($2,OFF,
            {
                line from C2.c to C1.c + (0, p/4)
            },
            {
                line from C2.c to C1.c
            }
        );
]')

move to (1,3);
swh(1, OFF);
swh(1, ON);
swh(2, OFF);
dot;
{ line up_   1; right_; S1: swh(2, OFF); "$S_1$" at S1.n; }
{ line down_ 1; right_; S2: swh(2, OFF); "$S_2$" at S2.n; }

,,,

cm_show('obr-327-1', sch, 600)
```



Obr. 11.1: Skript

Kapitola 12

Príloha 1. Použitie PyCirkuit v Jupyter-Notebooku

Pre špeciálne použitie môžeme vyvolať *PyCirkuit* aj z prostredia jupyter-notebooku, napr. ak potrebujeme v priamo notebooku vytvárať obrázky pomocou unifikovaného template.
Template sa nachádza v adresári *./img/* spolu s knižnicou doplnkových komponentov *user.ckt*. Pomocou funkcie *create_cm()* z pomocnéj knižnice *cm_utils* sa z template vygeneruje nový súbor s definíciou makier, tento je vhodné uložiť do adresáru *./img/* pre budúci export do LaTex-u.

```
from img.cm_utils import *
import os

create_cm('./img/obr-03.ckt', overwrite='No')
```

Editovanie zapojenia

Vytváranie zapojenia je pomocou externého editora *pycirkuit*, ktorý umožňuje náhľad schémy a jej export (nastavenie v konfigurácii) do obrázku *png*.

```
_ = os.system('pycirkuit ./img/obr-03.ckt &')
```

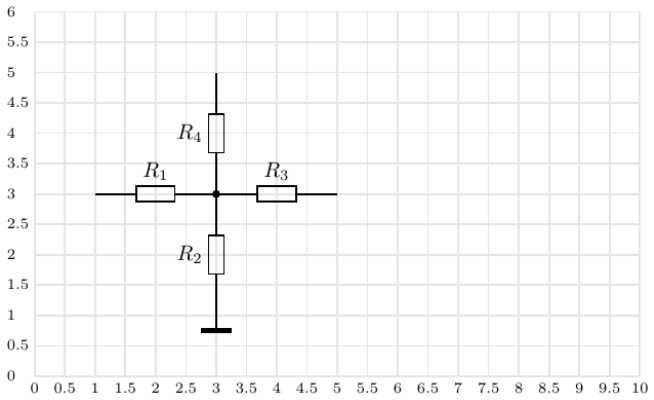
V PyCirkuit vytvoríme zapojenie

```
.PS
pi=3.14159265359
                        # parametre z PIC (resp. GNU PIC)
scale = 2.54            # cm - jednotka pre obrazok
maxpswid = 30           # rozmery obrazku
maxpsht = 30            # 30 x 30cm, default je 8.5x11 inch
cct_init                # inicializacia lokalnych premennych

arrowwid  = 0.127       # parametre sipok - sirka
arrowht = 0.254         # dlzka
include(user.ckt)       # externa kniznica
#-------------------------------------------------------------------------
Origin: Here
size_x = 10
size_y = 6
d = 2;                  # 1cm zakladna dlzka

# mriezka
move to (-0.5, size_y/2); grid(size_x, size_y); move to 0,0;
#=========================================================================
move to 1,3
```

```
R1: resistor(d, E); llabel(,R_1,); dot;
{
  right_;
  R3: resistor(d, E); llabel(,R_3,);
}

{
  up_;
  R4: resistor(d, E); llabel(,R_4,);
}

down_;
R2: resistor(d, E); rlabel(,R_2,);
gnd;

.PE
```

Obrázok vygenerovany pomocou PyCirkuit, zobrazenie je pomocou funkcie, pretože aktuálny stav obrázku vyžaduje refreš cache prehliadača.

```
show_png('./img/obr-03.png', 500)
```



Obr. 12.1: Zobrazenie vystupu z PyCirkuit