MAKE*CIRC*

A METAPOST library for electrical circuit diagrams drawing

Gustavo S. Bustamante Argañaraz[*]

January 9, 2004

Contents

1 Introduction

MAKE*CIRC* is a METAPOST library that contains diverse symbols (the majority of wich are electrician) for the use in circuit diagrams. MAKE*CIRC* tries to offer a high quality tool, with a simple syntax.

Although MAKE*CIRC* uses all power of METAPOST, it doesn't require the learning of this language to be used. However, it's knowledge makes easier the library implementation.

---

[*] Thanks to José Luis Díaz for the support given, answering my questions about METAPOST, and providing the `latex.mp` library, which is used in this library. Also thanks to Sebastián Pablo Sánchez for the help in this english version manual.

MAKE*CIRC* is completely integrated with LaTeX documents and with other METAPOST drawings/graphics. Its output is a POSTSCRIPT file.

2 Configuration

MAKE*CIRC* provides three parameters that can be configured externally:

- `linewd` specifies line thickness;
- `lbsep` specifies the separation between the label and the symbol;
- `dimen` specifies the symbols size; and
- `labeling` which can take two values, `rotatelabel` so that symbol's labels rotate together with them, or `norotatelabel` (default) so that labels stay in normal position.

The first three parameters should be scaled to modify their default value, that is, if you want line thickness to be the double of the current value, you should write:

```
linewd:=2linewd;
```

In the same way you should do with the other two:

```
lbsep:=2lbsep; dimen:=2dimen;
```

where `dimen` is a parameter (characteristic dimension) that varies according to which symbol it refers to, just as it is shown in table 1.

The `labeling` parameter, as was described, can take two values, which should be changed specifying `labeling:=value`, for example:

```
labeling:=rotatelabel;
```

The four configuration parameters can be applied whether to determined elements, or to all the circuit elements. In this case you should place them before beginning the symbols insertion.

Once `linewd` value was scaled a certain `factor`, to return it to its normal value you should write

```
linewd:=linewd/factor
```

because the `linewd` value is `factor` times `linewd`, therefore if you divide it by the same factor, the result will be `linewd`.

For example:

Table 1: Characteristic dimensions of symbols.

Element	**Characteristic dimension**
Resistor	`rstlth`
Capacitor	`platsep`
Inductor	`coil`
Transformer	
Source	`ssize`
Battery	
Motor	
Generator	
Measurement instrument	
Lamp	
Diode	`diodeht`
Transistor	`bjtlth`
Impedance	`implth`
Rheostat	`rheolth`
Ground	`gndlth`
Junction	`junctiondiam`
Switch	`ssep`

```
element1; element2;
labeling:=rotatelabel;
element3; % in this element the labels are rotated %
linewd:=2linewd;
element4; % in this element also, and in addition the
% line thickness is increased the double.
labeling:=norotatelabel; linewd:=linewd/2;
% here they came back to the default value %
element5; element6;
```

If you want to increase proportionally the size of all the circuit elements with the line thickness and labels size, you should use as last instruction

```
scalecirc factor;
```

with which the circuit size is scaled an amount `factor`.

3 Usage

The methodology that MAKE*CIRC* suggests for the creation of circuit diagrams is the following one:

a) to place the symbols in the wanted position; and
b) to connect them using their pins.

I think this method is more convenient, but you can implement any other that results simpler and more effective for you.

3.1 Begin

To begin, you should specify the library input in the following way

```
input makecirc;
```

Then, you should call `initlatex` function. This macro writes a LaTeX header in an auxiliary file.[1] By calling this function, you can specify additional packages and macros definitions to use with LaTeX.

```
        initlatex("\usepackage{package}" &
                  "\usepackage{other_package}" &
                  "\newcommand{command}{definition}");
```

The next step is to begin the circuit diagrams using the instruction `beginfig(n)`, being `n` the figure number. When you conclude the circuit, write `endfig`. If you want to make another circuit, you should write again:

```
        beginfig(k); % k is another number different of n %
        % ... here goes the circuit ... %
        endfig;
```

When you conclude all the circuits, write `end`.

It is important to mention that all the statements used in METAPOST —and therefore in MAKE*CIRC*— end with semicolon (`;`).

3.2 Symbols

In this section I will describe the syntax to compose the symbols in MAKE*CIRC*.

The majority of the most common symbols are composed in the following way:

```
element.α(z,type,angle,name,value);
```

---

[1] You don't need to call this macro if you use TeX as typesetting engine.

where: α is an alphabetical character (a, b, c, A, B, C, ab, cd, etc.) that will be used after as part of the element pins;
`z=(x,y)` are the insertion point coordinates of the element;
`type` is an element subcategory that will depend on which type of element is being inserted;
`angle` is the angle in degrees that the element will be rotated (counter-clockwise) starting from its initial position;
`name` is the name or characteristic letter of the element (`L`, `R`, etc.); and
`value` is the numeric value of the element.

However, there are other elements that contain more or less parameters that will be described along this section.

It should be added that the parameters `name` and `value` should be written between inverted commas (`"`). In addition LaTeX commands can be used in these parameters.

The `name` parameter is defined in *mathematical mode*, that is, in LaTeX language between $ and $. Therefore all mathematical functions and LaTeX commands are valid. This means, you can write `"L_1^2"` and the result will be L_1^2, `"v=\hat{V}\sin\omega t"` and the result will be $v = \hat{V}\sin\omega t$, etc. This is due in most cases the name of a symbol is a constant or mathematical variable (L, R, Z, v, i, etc.).

In contrast, the `value` parameter is defined in *text mode*, as is required in most cases. (`"10 V"` $\Longrightarrow$ 10 V, `"50 pF"` $\Longrightarrow$ 50 pF, etc.). An exception to this case —that require to be written in mathematical mode— is the electrical resistance unit (ohm Ω). To make easier the use of this package for those with no knowledge of LaTeX, **MAKE***CIRC* includes the `\ohm` and `\kohm` commands. Therefore, if you write `"10\ohm"`, the result will be 10 Ω, `"42\kohm"` and the result will be 42 kΩ. Note that there are no spaces between text and commands.

Also, **MAKE***CIRC* includes another command, `\modarg`, to write complex numbers in the notation module-argument.[2] For instance if you write `"\modarg{220}{30}"` the result will be $220\underline{/30^\circ}$.

It is important to mention that if labels' text contains errors, LaTeX will be unable to complete its compilation. This will make impossible to run METAPOST on the source file —even if the errors are corrected— until you delete the auxiliar file (which has `.mpt` extension).

Next will described and illustrated all the available symbols in **MAKE***CIRC*. It will be also indicated which is the pin wich is positioned on the `z` insertion point of the element.

Element: Inductor

Syntax: `inductor.`α`(z,type,angle,name,value)`

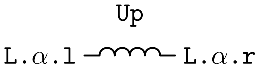
Available types: `Up|Down`

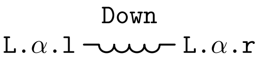
---

[2] Thanks to Javier Bezos for this.

Pins: `L.`α`.l|L.`α`.r`

Positioning pin in `z`: `L.`α`.l`





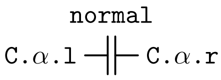
Element: Capacitor

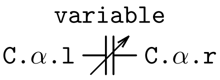
Syntax: `capacitor.`α`(z,type,angle,name,value)`

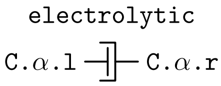
Available types: `normal|variable|electrolytic|variant`

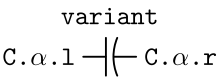
Pins: `C.`α`.l|C.`α`.r`

Positioning pin in `z`: `C.`α`.l`









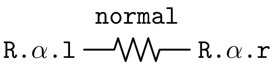
Element: Resistor

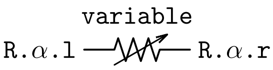
Syntax: `resistor.`α`(z,type,angle,name,value)`

Available types: `normal|variable`

Pins: `R.`α`.l|R.`α`.r`

Positioning pin in `z`: `R.`α`.l`





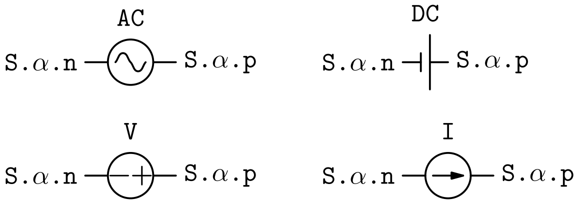
Element: Source

Syntax: `source.`α`(z,type,angle,name,value)`

Available types: `AC|DC|V|I`

Pins: `S.`α`.n|S.`α`.p`

Positioning pin in `z`: `S.`α`.n`



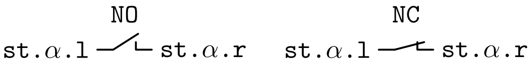
Element: Switch

Syntax: `switch.α(z,type,angle,name,value)`

Available types: `NO|NC`

Pins: `st.α.l|st.α.r`

Positioning pin in `z`: `st.α.l`

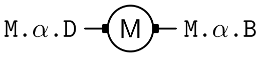


Element: Motor

Syntax: `motor.α(z,angle,name,value)`

Pins: `M.α.D|M.α.B`

Positioning pin in `z`: `M.α.D`

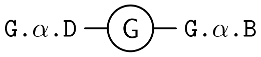


Element: Generator

Syntax: `generator.α(z,angle,name,value)`

Pins: `G.α.D|G.α.B`

Positioning pin in `z`: `G.α.D`

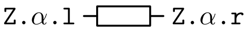


Element: Impedance

Syntax: `impedance.α(z,angle,name,value)`

Pins: `Z.α.l|Z.α.r`

Positioning pin in `z`: `Z.`α`.l`

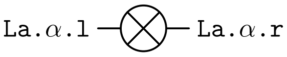


Element: Lamp

Syntax: `lamp.`α`(z,angle,name,value)`

Pins: `La.`α`.l|La.`α`.r`

Positioning pin in `z`: `La.`α`.l`

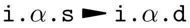


Element: Current

Syntax: `current.`α`(z,angle,name,value)`

Pins: `i.`α`.s|i.`α`.d`

Positioning pin in `z`: `i.`α`.s`

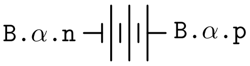


Element: Battery

Syntax: `battery.`α`(z,angle,name,value)`

Pins: `B.`α`.n|B.`α`.p`

Positioning pin in `z`: `B.`α`.n`



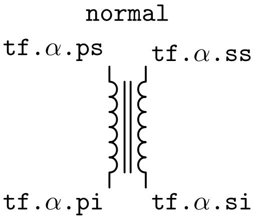
Element: Transformer

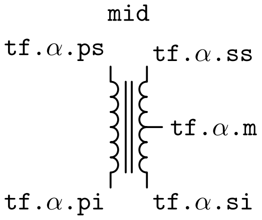
Syntax: `transformer.`α`(z,type,angle)`

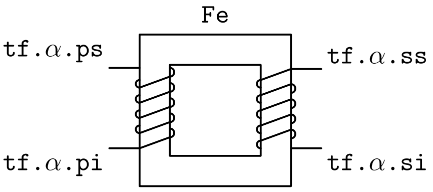
Available types: `normal|midpoint`

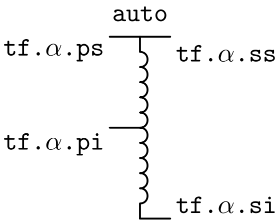
Pins: `tf.`α`.pi|tf.`α`.ps|tf.`α`.si|tf.`α`.ss|tf.`α`.m`

Positioning pin in `z`: `tf.`α`.pi`









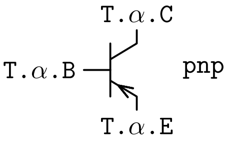
Element: Transistor

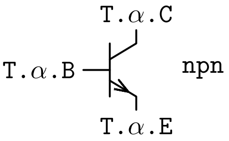
Syntax: `transistor.`α`(z,type,angle)`

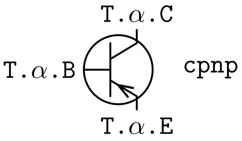
Available types: `pnp|npn|cpnp|cnpn`

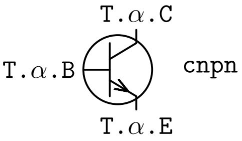
Pins: `T.`α`.B|T.`α`.C|T.`α`.E`

Positioning pin in `z`: `T.`α`.B`









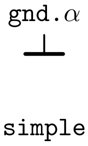
Element: Ground

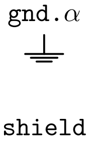
Syntax: `ground.`α`(z,type,angle)`

Available types: `simple|shield`

Pins: `gnd.`α

Positioning pin in `z`: `gnd.`α





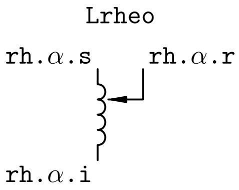
Element: Rheostat

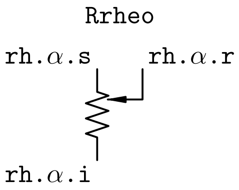
Syntax: `rheostat.`α`(z,type,angle)`

Available types: `Lrheo|Rrheo`

Pins: `rh.`α`.i|rh.`α`.s|rh.`α`.r`

Positioning pin in `z`: `rh.`α`.i`





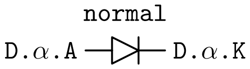
Element: Diode

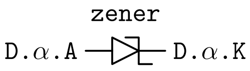
Syntax: `diode.`α`(z,type,angle,pin,name,value)`

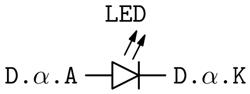
Available types: `normal|zener|LED`

Pins: `D.`α`.A|D.`α`.K`

Positioning pin in `z`: `D.`α`.A (pin=pinA)|D.`α`.K (pin=pinK)`







The diode is a special element, that is why it requires a special syntax. This is due any circuit that contains a diode behaves in different ways if the diode is in direct or inverse polarization. This means that is important the diode placement sense. For this reason, a `pin` parameter is added, which can take two values: `pinA` or `pinK`. This indicate what pin is placed on the `z` position, if the anode (`pinA`) or the cathode (`pinK`).

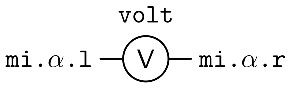
Element: Measurement instrument

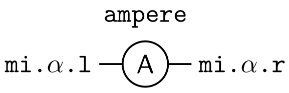
Syntax: `meains.`α`(z,type,angle,text)`

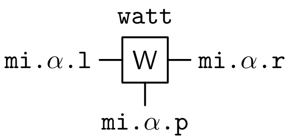
Available types: `volt|ampere|watt`

Pins: `mi.`α`.l|mi.`α`.r|mi.`α`.p`

Positioning pin in `z`: `mi.`α`.l`





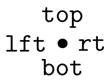


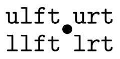
Element: Junction

Syntax: `junction.`α`(z,text)(pos)`

Position (pos): `top|bot|lft|rt|ulft|urt|llft|lrt`

Pins: `J.`α

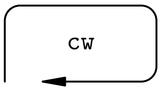


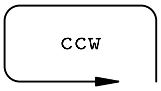


Element: Mesh current

Syntax: `imesh(center,width,height,sense,angle,name)`

Sense: `cw|ccw`





3.2.1 Positioning

You can place a symbol using relative coordinates (the most usual way) or absolute coordinates. The relative coordinates generally will refer to element pins placed previously.

The first symbol should be placed on the coordinates origin (only for simplicity) always using absolute coordinates. For example:

```
inductor.a(origin,Up,0,"L_1","10 H");
```

being `origin =(0,0)`.

The remaining elements are placed starting from this element using relative coordinates in the form `pin.ref+(x,y)`. This means that the next element will be placed a distance `x` to the right and `y` above, starting form the pin `pin.ref`. For example if the insertion point of my next symbol is `L.a.r+(1cm,5mm)`, it means that the symbol will be placed `1cm` to the right and `5mm` above the pin `L.a.r`, while the statement `L.a.l+(-1cm,5mm)` sets the point `1cm` to the left and `5mm` above the pin `L.a.l`.

However this doesn't cover all the user needs because you may want to center an element between two points (or pins) and you may not know their coordinates. For these cases MAKE*CIRC* provides the following statement:

```
centreof.α(pin1,pin2,sym);
```

being `sym` the symbol you want to center. To indicate this, you will use a convenient abbreviation of the elements (see table 2).

Table 2: Elements abbreviations.

Element	Abbreviation
Inductor	`ind`
Capacitor	`cap`
Motor	`mot`
Generator	`gen`
Transformer	`tra`
AC source	`sac`
DC source	`sdc`
Current source	`si`
Voltage source	`sv`
Battery	`bat`
Resistor	`res`
Diode	`dio`
Transistor	`bjt`
Measurement instruments	`ins`
Impedance	`imp`
Lamp	`lam`
Switch	`swt`
Current	`cur`

This statement returns the element insertion point (`c.α`) and their angle `phi.α` whose direction is the straight line that unites the pins `pin1` and `pin2`.

To clear these aspects of positioning, an example will be given. Suppose you want to include a resistor R of $10\,\Omega$ rotated 90 degrees in the coordinates origin. Then you write (see section 3.2):

```
resistor.a(origin,normal,90,"R","10\ohm");
```

Next you include an inductor L at 2 centimeters at right of the resistor by writing:

```
inductor.a(R.a.r+(2cm,0),Up,-90,"L","");
```

Now if you want to center a capacitor between these two elements, you should write:

```
centreof.A(R.a.r,L.a.l,cap);
```

which returns the insertion point `c.A` where the capacitor must be placed, and the angle `phi.A` that it will be rotated. With these data you place the capacitor C in the following way:

```
capacitor.a(c.A,normal,phi.A,"C","");
```

The complete code would be:

```
beginfig(1);
        resistor.a(origin,normal,90,"R","10\ohm");
        inductor.a(R.a.r+(2cm,0),Up,-90,"L","");
        centreof.A(R.a.r,L.a.l,cap);
        capacitor.a(c.A,normal,phi.A,"C","");
endfig;
```

whose result is shown in figure 1.

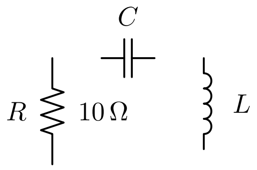


Figure 1: Example of positioning.

Another possibility that can be present in the moment of positioning an element is that you want to center a symbol under or next to another. For this, MAKE*CIRC* provides the following statement:

```
centerto.α(pin1,pin2,dist,sym)
```

being `dist` the separation distance from the element whose pins are `pin1` and `pin2` to the element you want to place. This distance can be a positive value —if you want to move the element to the right or above— or negative —if you want to move it to the left or under—. For example, using the previous case, you can see that the coil isn't centered with the resistor. To do this, you should proceed in the following way:

```
beginfig(2);
        resistor.a(origin,normal,90,"R","10\ohm");
        centerto.A(R.a.l,R.a.r,2cm,ind);
        inductor.a(A,Up,90,"L","");
endfig;
```

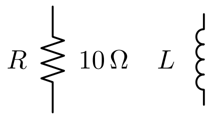


Figure 2: Elements centering example.

The result of entering this code is shown in figure 2.

If you now want, as in the previous case, to center the capacitor between these two element just as they are, the result will be the shown in figure 3, that is, the capacitor is slightly rotated in its right part. This is due to the pins used for centering have different heights, therefore the `phi` angle of the `centreof` statement, will have the direction of the straight line that unites these two points.

Source code of figure 3

```
beginfig(3);
        resistor.a(origin,normal,90,"R","10\ohm");
        centerto.A(R.a.l,R.a.r,2cm,ind);
        inductor.a(A,Up,90,"L","");
        centreof.B(R.a.r,L.a.r,cap);
        capacitor.a(c.B,normal,phi.B,"C","");
endfig;
```

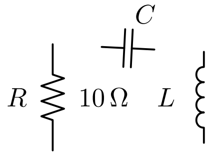


Figure 3: Rotated capacitor for being centred among two pins which heights are different.

You can specify the angle zero (0) instead of the angle `phi` that returns the `centreof` statement and the capacitor will be correctly positioned (horizontally), but this will generate problems later when you make the connection of the elements. Instead of this you should use the statements `xpart` and `ypart`, that are characteristic of METAPOST and not of **MAKE***CIRC*.

These statements specify the horizontal (`xpart`) or vertical (`ypart`) absolute coordinate of a point. In our case, it's necessary to know the coordinates of the point z_1 (see figure 4), which is at the same height that the resistor pin, and therefore it will allow to make a correct centering of the capacitor. The coordinates of this point are not exlplicitly known, but they are implicit, because the z_1 point has the same x coordinate (horizontal) that the pin `L.a.r` and the same y coordinate (vertical) that the point `R.a.r`, that is

$$z_1 = (\texttt{xpart L.a.r}, \texttt{ypart R.a.r})$$

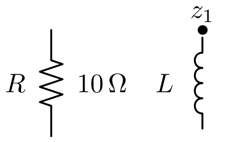


Figure 4: Point z_1 at the same height that the resistance pin.

This is the point you should use as `pin2` for centering, this means that the line 5 of the source code in figure 3 should be replaced by the following:

```
centreof.B(R.a.r,(xpart L.a.r,ypart R.a.r),cap);
```

Also it can be defined a point `z1` with these coordinates and then specify it in the centering,[3] that is:

```
z1=(xpart L.a.r,ypart R.a.r);
centreof.B(R.a.r,z1,cap);
```

If you use this last method, the corrected source code, which result is shown in figure 5 is the following:

Source code of figure 5

```
beginfig(5);
        resistor.a(origin,normal,90,"R","10\ohm");
        centerto.A(R.a.l,R.a.r,2cm,ind);
        inductor.a(A,Up,90,"L","");
        z1=(xpart L.a.r,ypart R.a.r);
        centreof.B(R.a.r,z1,cap);
        capacitor.a(c.B,normal,phi.B,"C","");
endfig;
```

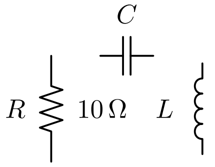


Figure 5: Use of statements `xpart` and `ypart` to center correctly the capacitor.

3.2.2 Wiring

To wire the elements with each other, you use two statements:

---

[3] This is also characteristic of METAPOST.

`wire(pin1,pin2,type); and wireU(pin1,pin2,dist,type);`

Available types for these statements are shown in table 3.

Table 3: Available types for the wire statements.

Statement	Available types
`wire`	`nsq|udsq |rlsq`
`wireU`	`udsq|rlsq`

If the pins are at the same height, you should specify `nsq` type, with which the wired will be in straight line, from `pin1` to `pin2`. This is also valid for any wire that you want to make in straight line. If the pins, on the other hand, are at different heights you should specify `udsq` type, with which the wire will be made with straight angles, starting from `pin1` to above or below and then to `pin2`, or `rlsq` and the wire will do in squares but starting from `pin1` to right or left and then to `pin2`.

These last two are also valid for `wireU` statement, but in this, is added one parameter, because you should use this statement when the wire need three lines (in `U` form), that is when you have to connect two branches in parallel. In this case, the direction up, down, right or left is given by the sign of the parameter `dist` (see table 4).

Table 4: Direction according to the `dist` sign.

Type	Sign of `dist`	
	+	−
`udsq`	up	down
`rlsq`	right	left

To illustrate this, the previous examples will be taken into account and will be wired. For the example of figure 1 you will use three wire statements: one to connect the resistance with the capacitor, other to connect the capacitor with the coil, and another to connect the coil with the resistance. For the first one and second, as the pins are at the same height it will be:

```
wire(R.a.r,C.a.l,nsq);
wire(C.a.r,L.a.l,nsq);
```

and for the third, as the pins aren't at the same height two different wiring possibilities exists:

if you start from the coil you should use the `udsq` type because the coil pin is above that of the resistance, therefore first it will make a line down and then to the resistance pin; on the other hand

if you start from the resistance you should use `rlsq` type because the resistance pin is lower than the coil pin, therefore it will first make a line toward the right and then to the coil pin.

```
wire(L.a.r,R.a.l,udsq);
% o bien %
wire(R.a.l,L.a.r,rlsq);
```

Both statements give the same result.

Adding the connection lines to the source code of figure 1 you obtain the following complete code, which result is shown in figure 6.

```
beginfig(6);
        resistor.a(origin,normal,90,"R","10\ohm");
        inductor.a(R.a.r+(2cm,0),Up,-90,"L","");
        centreof.A(R.a.r,L.a.l,cap);
        capacitor.a(c.A,normal,phi.A,"C","");
        wire(R.a.r,C.a.l,nsq);
        wire(C.a.r,L.a.l,nsq);
        wire(L.a.r,R.a.l,udsq);
        % or
        % wire(R.a.l,L.a.r,rlsq);
endfig;
```

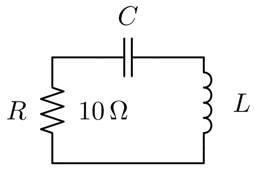


Figure 6: Example of elements wiring.

If the example of figure 5 is used, the wiring will be:

```
wire(R.a.r,C.a.l,nsq);
% resistance and capacitor, pins at the same height %
wire(C.a.r,L.a.r,rlsq);
% capacitor and coil, pin C.a.r above L.a.r %
wire(L.a.l,R.a.l,udsq);
% coil and resistance, pin L.a.l above R.a.l %
```

Adding these lines in the source code of figure 5 result:

Source code of figure 7

```
beginfig(7);
        resistor.a(origin,normal,90,"R","10\ohm");
        centerto.A(R.a.l,R.a.r,2cm,ind);
        inductor.a(A,Up,90,"L","");
        z1=(xpart L.a.r,ypart R.a.r);
        centreof.B(R.a.r,z1,cap);
        capacitor.a(c.B,normal,phi.B,"C","");
        wire(R.a.r,C.a.l,nsq);
```

```
 9          % resistance and capacitor, pins at the same height %
10          wire(C.a.r,L.a.r,rlsq);
11          % capacitor and coil, pin C.a.r above L.a.r %
12          wire(L.a.l,R.a.l,udsq);
13          % coil and resistance, pin L.a.l above R.a.l %
14  endfig;
```

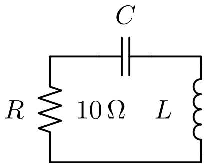


Figure 7: Other connection example.

To show the utility of `wireU` statement the source code of figure 7 will be used and the line 7 will be changed by the following:

```
capacitor.a(c.B+(0,5mm),normal,phi.B,"C","");
```

that is, the capacitor isn't placed in the point `c.B`; it's placed 5 millimeters above it (see section 3.2.1). Making this you should also change the line 8 because the resistance and capacitor pins are now at different heights:

```
wire(R.a.r,C.a.l,udsq);
```

Furthermore, if you want the circuit to be symmetrical, in the line 12 you can't use the `wire` statement because you first need to lower the line 5 millimeters (the same that the capacitor rose) and after that go towards the coil pin. Therefore, you will do:

```
wireU(L.a.l,R.a.l,-5mm,udsq);
```

Adding these lines in the source code results:

Source code of figure 8

```
beginfig(8);
        resistor.a(origin,normal,90,"R","10\ohm");
        centerto.A(R.a.l,R.a.r,2cm,ind);
        inductor.a(A,Up,90,"L","");
        z1=(xpart L.a.r,ypart R.a.r);
        centreof.B(R.a.r,z1,cap);
        capacitor.a(c.B+(0,5mm),normal,phi.B,"C","");
        wire(R.a.r,C.a.l,udsq);
        % resistance and capacitor, pins at the same height %
        wire(C.a.r,L.a.r,rlsq);
        % capacitor and coil, pin C.a.r above L.a.r %
        wire(L.a.l,R.a.l,-5mm,udsq);
        % coil and resistance, pin L.a.l above R.a.l %
endfig;
```

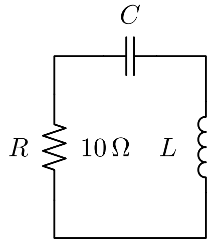


Figure 8: Connection example using `wireU`.

3.3 Adding of text to the circuits

In occasions it is necessary to have a tool that allows you to add text in the circuits because sometimes the information you can include in the symbol's labels is not enough. For this purpose MAKE*CIRC* provides two statements:

`puttext`⟨*pos*⟩`(text,z);` and `ctext`⟨*pos*⟩`(pin1,pin2,text,type);`

being `z` the point where the text will be placed and ⟨*pos*⟩ the position that the text will assume regarding `z`:

⟨*pos*⟩ → ⟨*`empty`*⟩|`top`|`bot`|`lft`|`rt`|`ulft`|`urt`|`llft`|`lrt`

If you let the ⟨*pos*⟩ argument empty, the text will be placed exactly in `z`, else it will be placed above (`top`), under (`bot`), left (`lft`), right (`rt`), above left (`ulft`), above right (`urt`), under left (`llft`) or under right (`lrt`) regarding `z`.

The `ctext` statement is for centering text between the pins `pin1` and `pin2`. The available types are: `witharrow` to draw a double arrow from `pin1` to `pin2`, and `noarrow` for only centering the text between these two points.

In both statements the text should be placed between inverted commas (").

To show an example, a transformer will be drawn and the labels V_1 and V_2 will be placed in the primary and secondary respectly. Also the number of spires N_1 in the primary winding and N_2 in the secondary winding will be placed (see figure 9).

Source code of figure 9

```
beginfig(9);
        transformer.a(origin,normal,0);

        junction.a(tf.a.ps-(1cm,0),"1a")(top);
        junction.b(tf.a.pi-(1cm,0),"1b")(bot);
        junction.c(tf.a.ss+(1cm,0),"2a")(top);
        junction.d(tf.a.si+(1cm,0),"2b")(bot);

        wire(tf.a.ps,J.a,nsq);
        wire(tf.a.pi,J.b,nsq);
        wire(tf.a.ss,J.c,nsq);
        wire(tf.a.si,J.d,nsq);

        puttext.bot("$N_1$",tf.a.pi);
        puttext.bot("$N_2$",tf.a.si);
        ctext.lft(J.a,J.b,"$V_1$",witharrow);
        ctext(J.c,J.d,"$V_2$",noarrow);
endfig;
```

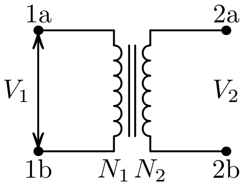


Figure 9: Transformer with text.

4 Compilation

When all work is done, you should compile the file to generate and visualize the results. For that, the first thing you should do is to save the file which will have an extension `mp`, for example: `circuits.mp`.

Then you should compile **twice** through METAPOST to make sure that the labels composed by LaTeX come out correctly. You will do this by writing in a command window (MSDOS in Windows) the following:

```
mp circuits
mp circuits
```

This will create files with the same name that the `mp` file but with numeric extension: `circuits.1, circuits.2,..., circuits.n`, being `n` the `beginfig` number (see section 3.1).

Now you only need to include the files `.n` in a LaTeX document. You will save this file with the name `circuits.tex`:

Minimal LaTeX document to visualize the results

```
\documentclass{article}
\usepackage{graphicx} % package for graphic inclusion %

\begin{document}
\centering
\includegraphics{circuits.1}
\includegraphics{circuits.2}
⋮
```

```
\includegraphics{circuits.n}
\end{document}
```

and compile it with LaTeX and `dvips` to obtain the final POSTSCRIPT file:

```
latex circuits
dvips circuits
```

Now you can visualize the file `circuits.ps` with Ghostview.

Other option is, if you only want to see the circuits, to use the following:[4]

```
tex mproof circuits.1 circuits.2 circuits.3
dvips mproof
```

and then visualize with Ghosview the file `mproof.ps`.

---

[4] Generally the file `mproof.tex` is included with METAPOST in the distributions.