

# VAUCANSON-G

## A package for drawing automata and graphs

User's manual

*Sylvain Lombardy*

IGM-LabInfo , Univ. Paris-Est

*Jacques Sakarovitch*

LTCI, CNRS / TELECOM ParisTech

June 01., 2026

Version 0.5

— o —

VAUCANSON-G is a package that allows to draw automata within texts written using L<sup>A</sup>T<sub>E</sub>X. It is a set of macros that uses commands of the beautiful PSTricks package, due to Timothy van Zandt<sup>1</sup> and which is part of the current L<sup>A</sup>T<sub>E</sub>X distribution. VAUCANSON-G is the result of the experience gained in composing the several hundreds of figures in the book *Eléments de théorie des automates* ([5]) and in several other papers and conferences slides of the authors.

The design of VAUCANSON-G implements the following underlying philosophy:

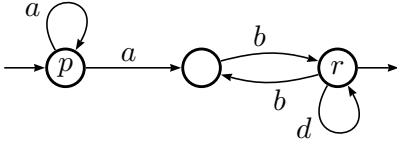
1. ‘simple’ automata should be described with simple commands.
2. Every element of a figure can easily be given a style (which differs from the implicit one).
3. The complexity of commands (or the number of things to be remembered to use them) should gradually grow with the complexity of the figure composed by these commands.
4. It should be possible to handle the figures both in size and appearance without modifying them.

The following example<sup>2</sup> shows how a simple automaton can be drawn with commands, in which only the minimal information needed (position and label of states, shape and label of transitions) is made explicit.

---

<sup>1</sup>The maintenance and evolution of PSTricks has now been taken over by H. Voss.

<sup>2</sup>somewhat artificial but which exhibits most of the simple commands.



```
\begin{VCPicture}{(0,-2)(6,2)}
% states
\State[p]{(0,0)}{A} \State{(3,0)}{B} \State[r]{(6,0)}{C}
% initial--final
\Initial{A} \Final{C}
% transitions
\EdgeL{A}{B}{a} \ArcL{B}{C}{b} \ArcL{C}{B}{b}
\LoopN{A}{a} \LoopS{C}{d}
%
\end{VCPicture}
```

The objectives of VAUCANSON-G are achieved by the implicit definition of a large number of parameters that control the geometry of the figure: size of states, width of lines, *etc.* and by the definition of commands that allow to handle and modify these parameters.

The following pages gives a presentation of VAUCANSON-G that gradually introduces the commands from the basic ones that allow to draw already a good deal of automata to the more sophisticated ones that are used to modify the internal parameters.

## Contents

<b>0</b>	<b>Downloading and using the package</b>	<b>3</b>
<b>1</b>	<b>Basic commands</b>	<b>4</b>
<b>2</b>	<b>Using preset parameters</b>	<b>7</b>
<b>3</b>	<b>Composing a picture</b>	<b>15</b>
<b>4</b>	<b>Parametrized drawing commands</b>	<b>18</b>
<b>5</b>	<b>Modifying and setting the parameters</b>	<b>22</b>
<b>6</b>	<b>Using “plain” PSTricks commands</b>	<b>30</b>
<b>A</b>	<b>Version history</b>	<b>31</b>
<b>B</b>	<b>Style files</b>	<b>32</b>

## 0 Downloading and using the package

The VAUCANSON-G package is downloadable from the URL:

`/igm.univ-mlv.fr/~lombardy/Vaucanson-G/`

and consists, beside the present manual, in seven files: the wrapper `vaucanson-g.sty`, the main macro file `VauCanSon-G.tex`, and five other ancillary files<sup>3</sup> for options, patches and colors: `VCPref-default.tex`, `VCPref-slides.tex`, `VCPref-beamer.tex`, `VCPref-mystyle.tex`, `VCColor-names.def` that should all be stored in the user's `texinputs` directory.

A  $\text{\LaTeX}$  file that will use VAUCANSON-G macros should contain the call to the wrapper in the preamble:

`\usepackage{vaucanson-g}`

The VAUCANSON-G commands are to be used in a  $\text{\LaTeX}$  file and not in a plain  $\text{\TeX}$  file for it uses the macro `\newcommand` that does not exist in  $\text{\TeX}$ .

VAUCANSON-G is based on the `PSTricks` package. The knowledge of this macro package, described in [1], [3, Ch. 4], [4], [6], is not necessary to use the VAUCANSON-G commands.<sup>4</sup> On the contrary, VAUCANSON-G has somehow be written to save on it. In particular, the VAUCANSON-G commands follow the simple (and rigid) syntax of  $\text{\LaTeX}$  commands concerning the way the arguments are specified, and not the much more flexible syntax of `PSTricks` itself.

But the call to `PSTricks` commands has the consequence that VAUCANSON-G commands generate pieces of `Postscript` files which in turn has two main outcomes<sup>5</sup>:

- VAUCANSON-G is not compatible with `pdflatex` ;
- the figures produced with VAUCANSON-G are not likely to be seen on a `DVI` viewer: the space occupied by the figure will appear empty and, in most cases, the labels will appear at the bottom of it.

VAUCANSON-G can be used with the `beamer` document class to design slides. The document class should be called with `xcolor=pst,dvips` options:

`\documentclass[xcolor=pst,dvips]{beamer}.`

Thus equipped, we may begin.

---

<sup>3</sup>*Cf.* Section 5 for an explanation on the role of these files.

<sup>4</sup>It probably helps in understanding how they are built, and is useful if one wants to make figures that consists of automata but also contains other graphical elements (*cf.* Section 6).

<sup>5</sup>which a user should have in mind before sending complaint and request.

## 1 Basic commands

An automaton is a *labelled directed graph* where some of the vertices, called *states*, are distinguished to be *initial* or *final* states. The basic commands allow then *to define* and *to draw* states, to give them the quality of being initial or final and to give them *labels* as well. They allow to draw *transitions* between states (that have been defined before); there are 3 kinds of transitions: (*straight*) *edges*, *arcs* and *loops*.

These commands have to be put in an environment that defines a *picture*. All graphical parameters of these elements (size, line width, *etc.*) are kept implicit when using these basic commands.

### 1.1 The VCPicture environment

<code>\begin{VCPicture}{(x_0, y_0)(x_1, y_1)}</code>	Defines a box that “encloses” the elements defined in the en-
<code>...</code>	vironment. Its lower left (resp. upper right) corner has coord-
<code>\end{VCPicture}</code>	inates $(x_0, y_0)$ (resp. $(x_1, y_1)$ ).

The  $x_i$  and  $y_i$  are either *length* or *numbers* in which case they represent the corresponding length in centimeter (which is the implicit length unit used in VAUCANSON-G). The length of the box is thus  $x_1 - x_0$ , its height  $y_1 - y_0$ , its *baseline point* stands at the middle of the height. As usual in T<sub>E</sub>X, a picture may well “contain” an element that actually lays outside of the box.

Note that the environment has only *one* argument consisting of *two* point coordinates enclosed in adjacent parentheses.

### 1.2 The State commands

States are represented by circles, inside which an information (the *Label*) can be written. Every state is given a *Name* which is used then internally in the figure to describe the transitions that start from it or end at it.

<code>\State[Label]{(x,y){Name}}</code>	Defines a state with name <i>Name</i> and draws it at the coordinate $(x, y)$ with label <i>Label</i> written inside.
---	---

The parameter *Label* is optional; default value is the empty string. It is written in ‘T<sub>E</sub>X mathmode’.

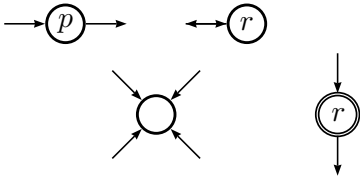
<code>\Initial[Dir]{Name}</code>	Makes the state <i>Name</i> initial, <i>i.e.</i> draws an incoming arrow from the direction <i>Dir</i> to state <i>Name</i> .
<code>\Final[Dir]{Name}</code>	Makes the state <i>Name</i> final, <i>i.e.</i> draws an outgoing arrow from state <i>Name</i> in the direction <i>Dir</i> .

The parameter *Dir* is optional; the possible values are the eight main compass points: *n*, *s*, *e*, *w*, *ne*, *nw*, *se* and *sw*. Default value is *w* for `\Initial`, *e* for `\Final`.

Many authors, in particular those dealing with automata on infinite words, rather indicate that a state is final by drawing it with doubleline. The reason why it is not possible to achieve this result by a sequence of two commands comes from the `PSTricks` macros that are called by the `VAUCANSON-G` commands.

`\FinalState[Label]{(x,y)}{Name}`

Same as `\State` but draws state *Name* with a doubleline which is a common way to indicate that a state is final.



```
\begin{VCPicture}{(0,-2)(6,2)}
% states
\State[p]{(0,1)}{A} \State{(2,-1)}{B} \State[r]{(4,1)}{C}
\FinalState[r]{(6,-1)}{D}
% initial--final
\InitialL{A} \Final{A}
\Initial[ne]{B} \Initial[nw]{B}
\Initial[se]{B} \Initial[sw]{B}
\Initial{C} \Final[w]{C} \Initial[n]{D} \Final[s]{D}
%
\end{VCPicture}%
```

### 1.3 The Edge commands

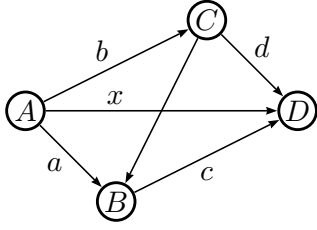
Edges, like the other transitions, are *oriented* from the origin state to the end state. The *left* and the *right* sides of a transition are defined with respect to this orientation, being the left and the right hand of someone standing on the origin state and following the transition towards the end state.

`\EdgeL[Pos]{Start}{End}{Label}`

Draws a straight transition from the state *Start* to the state *End* with the label *Label* on the left side for `\EdgeL`, on the right side for `\EdgeR`.

`\EdgeR[Pos]{Start}{End}{Label}`

The parameter *Pos* is optional. It is a number ( $0 \leq Pos \leq 1$ ) that defines the position of the label on the edge. Default value is .45 and may be used most of the time — this slight disymmetry gives some “dynamic” to the figure. Explicit value for *Pos* is used to avoid collision with other elements of the figure. As for states, the label is written in ‘`TEX` mathmode’.



```
\begin{VCPicture}{(0,-2)(6,2)}
% states
\State[A]{(0,0)}{A} \State[B]{(2,-2)}{B} \State[C]{(4,2)}{C}
\State[D]{(6,0)}{D}
% transitions
\EdgeR{A}{B}{a} \EdgeL{A}{C}{b} \EdgeR{B}{D}{c}
\EdgeL{C}{D}{d} \EdgeL{C}{B}{c} \EdgeL[.3]{A}{D}{x}
%
\end{VCPicture}
```

## 1.4 The Arc commands

Arcs are transitions that link the *Start* state to the *End* state with a curved line. The main parameter of such a curve is the *angle* made at both ends by the curve with the straight line that goes from *Start* to *End* and it can be oriented to the *left* or to the *right*. In the sequel, the letter *X* stands for either L or R.

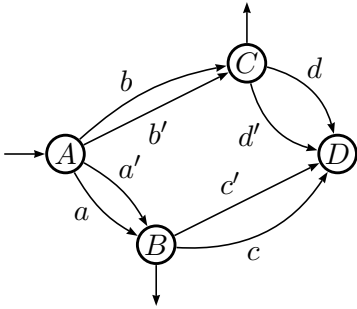
`\ArcX[Pos]{Start}{End}{Label}`

`\LArcX[Pos]{Start}{End}{Label}`

Same syntax as `\EdgeX`.

The starting angle is  $15^\circ$  for `ArcX`,  $30^\circ$  for `LArcX`.

Parameter *Pos* is optional. Default value is .4 .



```
\begin{VCPicture}{(0,-2)(6,2)}
% states
\State[A]{(0,0)}{A} \State[B]{(2,-2)}{B} \State[C]{(4,2)}{C}
\State[D]{(6,0)}{D}
%initial-final
\Initial{A} \Final[s]{B} \Final[n]{C}
% transitions
\ArcR{A}{B}{a} \ArcL{A}{C}{b} \LArcR{B}{D}{c} \LArcL{C}{D}{d}
\ArcL{A}{B}{a'} \EdgeR{A}{C}{b'} \EdgeL{B}{D}{c'}
\LArcR{C}{D}{d'}
%
\end{VCPicture}
```

## 1.5 The Loop commands

Loops are transitions that link a state to itself, a state which is thus both the origin and the end of the transition. The two main parameters of such a loop are the *opening angle* and the *direction* of the loop. In the sequel, the letter *Y* stands for any of the compass points: N, S, E, W, NE, NW, SE or SW.

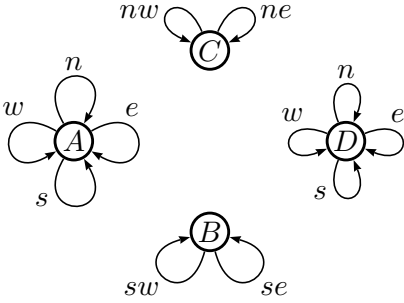
`\LoopY[Pos]{Name}{Label}`

`\CLoopY[Pos]{Name}{Label}`

Draw a loop around the state *Name* with label *Label*.

The direction of the loop is given by *Y*. The opening of the loop is  $60^\circ$  for the `LoopY` commands,  $44^\circ$  for the `CLoopY` commands.

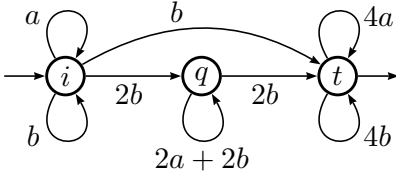
Parameter *Pos* is optional. Default value is .25 .



```
\begin{VCPicture}{(0,-3)(7,3)}
% states
\State[A]{(0,0)}{A} \State[B]{(3,-2)}{B} \State[C]{(3,2)}{C}
\State[D]{(6,0)}{D}
% transitions
\LoopE{A}{e} \LoopW{A}{w} \LoopS{A}{s} \LoopN[.5]{A}{n}
\LoopSE[.5]{B}{se} \LoopSW[.5]{B}{sw}
\CLoopNE[.6]{C}{ne} \CLoopNW[.6]{C}{nw}
\CLoopE{D}{e} \CLoopW{D}{w} \CLoopS{D}{s} \CLoopN[.5]{D}{n}
%
\end{VCPicture}
```

## 1.6 An example

This is an automaton with multiplicity in  $\mathbb{N}$ . The multiplicity of a word  $u$  is the square of the number represented by  $u$  in base 2, when  $a$  is interpreted as 0 and  $b$  as 1.



```
\begin{VCPicture}{(0,-2)(6,2)}
% states
\State[i]{(0,0)}{A} \State[q]{(3,0)}{B} \State[t]{(6,0)}{C}
% initial-final
\Initial{A} \Final{C}
% transitions
\EdgeR{A}{B}{2b} \EdgeR{B}{C}{2b} \LArcL{A}{C}{b}
\LoopN{A}{a} \LoopS{A}{b} \LoopS[.5]{B}{2a+2b}
\LoopN[.75]{C}{4a} \LoopS[.75]{C}{4b}
%
\end{VCPicture}
```

## 2 Using preset parameters

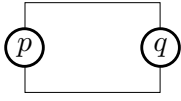
Together with the basic commands,  $\overline{\text{VAUCANSON-G}}$  gives easy access to two convenient parameters for drawing automata: global scaling and state size.

### 2.1 Global scaling

The meticulous reader has noticed that the actual size, mesured for instance by the distance between the states, is not the one given in the source code. This is because we have slightly cheated and the source code of the picture is indeed encapsulated in a `\VCDraw` command that uses a preset parameter `\VCScale` as shown below, with no cheating anymore.



```
\begin{VCPicture}{(0,-1)(3,1)}
\State[p]{(0,0)}{A} \State[q]{(3,0)}{B}
\end{VCPicture}
```



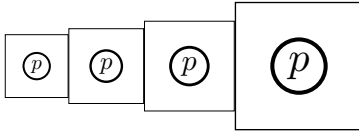
```
\VCDraw{%
\begin{VCPicture}{(0,-1)(3,1)}
\State[p]{(0,0)}{A} \State[q]{(3,0)}{B}%
\end{VCPicture}}
```

Frames that appear on figures below are an option of VAUCANSON-G that we shall see later. They point out the effect of the scale.

There is no reason indeed for an author to “program” the drawing of an automaton at the size it will appear on his, or her, paper. On the contrary, he, or she, is likely to sketch the figure on a ruled paper at a larger scale, giving the points to be defined integer coordinates. And then she, or he, will try to put the figure within the text at a reasonable size and thus would like to scale it.

The parameter `\VCScale` contains this global scaling factor and is initialized to the value 0.6 , which corresponds to the command `\MediumPicture`. Three other commands: `\TinyPicture` , `\SmallPicture` and `\LargePicture` are available, that set `\VCScale` to the values 0.42 , 0.5 and 0.85 respectively.

These command have to be used *outside* the scope of the `VCPicture` environment; and this environment should be called as the parameter of a `\VCDraw` .

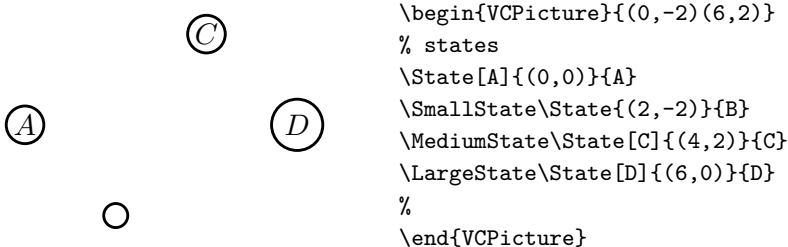


```
\TinyPicture\VCDraw{%
\begin{VCPicture}{(-1,-1)(1,1)}\State[p]{(0,0)}{A}\end{VCPicture}}
\SmallPicture\VCDraw{%
\begin{VCPicture}{(-1,-1)(1,1)}\State[p]{(0,0)}{A}\end{VCPicture}}
\MediumPicture\VCDraw{%
\begin{VCPicture}{(-1,-1)(1,1)}\State[p]{(0,0)}{A}\end{VCPicture}}
\LargePicture\VCDraw{%
\begin{VCPicture}{(-1,-1)(1,1)}\State[p]{(0,0)}{A}\end{VCPicture}}
```

In the sequel, and unless otherwise stated, all the figure will be given at the `\MediumPicture` scale without mentioning the command `\VCDraw` in the source code.

## 2.2 State size

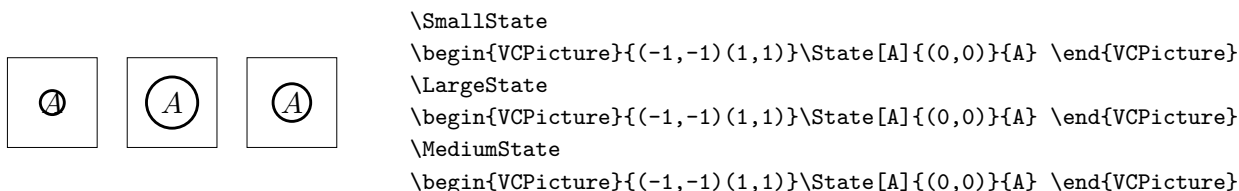
There are *three preset sizes* for the states, that are called by the commands `\SmallState`, `\MediumState` and `\LargeState`, that call for states of diameter 0.6cm, 0.9cm and 1.2cm respectively. The implicit size, *i.e.* the one that states have if no explicit command is given, is `\MediumState`. The “small” states are not likely to receive labels, unless their size are changed by other commands.



```
\begin{VCPicture}{(0,-2)(6,2)}
% states
\State[A]{(0,0)}{A}
\SmallState\State[B]{(2,-2)}{B}
\MediumState\State[C]{(4,2)}{C}
\LargeState\State[D]{(6,0)}{D}
%
\end{VCPicture}}
```



When called outside the scope of the environment `\VCPicture` the commands `\SmallState`, `\MediumState` and `\LargeState` set the implicit size of the states in all pictures from that point on.



## 2.3 Variable size states

One can draw states which will be called *variable states* and whose *width* is variable and *fits the width of the label*.<sup>6</sup> The height of these variable states is fixed by the current preset state size.

`\StateVar[Label]{(x,y)}{Name}` Same as syntax as `\State`.

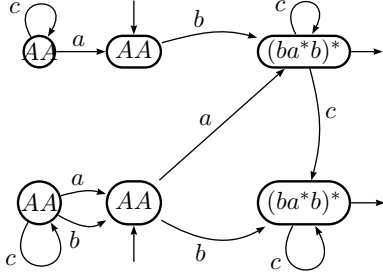
Note that the current version of  $\overline{\text{VAUCANSON-G}}$  uses a trick to make these states with adaptable width. They look like Egyptian cartouches but they are indeed *rectangles* which can be observed when transitions reach these states in an oblique way.

Arcs between variable states of different width will look asymmetric and often ungainly. Other loops than “north” and “south” loops should not be drawn around variable states. Moreover, in order to draw loops that have the same appearance as loops around standard states, one should use `\LoopVarN` or `\LoopVarS` that have the same syntax as `\LoopN` or `\LoopS`. Another possibility is to call the `\VarLoopOn` command to make all the subsequent loops suited to variable states. Switching back to ‘normal’ loops is made by `\VarLoopOff`.

For the same reason, it is *not possible* to make a variable state initial or final with diagonal arrows (*ne*, *nw*, *se*, *sw*).

---

<sup>6</sup>This makes sense for long labels and it may be the case then that the label is not a mathematical variable or expression any more but simple text. In order to write labels in  $\text{\TeX}$  LR mode, the easiest way is to write it as the parameter of a `\text` command, provided the `amsmath` package has been called in the preamble.

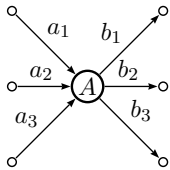


```
\SmallPicture\VCDraw{\begin{VCPicture}{(0,-3)(8,3)}
% states
\State[AA]{(0,2)}{A} \StateVar[AA]{(2.5,2)}{B}
\StateVar[(b a^* b)^*]{(7,2)}{C}
\LargeState \StateVar[(b a^* b)^*]{(7,-2)}{C1}
\State[AA]{(0,-2)}{A1} \StateVar[AA]{(2.5,-2)}{B1}
%
\Initial[n]{B} \Final{C} \Initial[s]{B1} \Final{C1}
%
\EdgeL{A}{B}{a} \ArcL{B}{C}{b} \ArcL{C}{C1}{c} \EdgeL{B1}{C}{a}
\ArcL{A1}{B1}{a} \LArcR{A1}{B1}{b} \LArcR{B1}{C1}{b}
\LoopN{A}{c} \LoopVarN{C}{c}
\LoopS{A1}{c} \VarLoopOn\LoopS{C1}{c}\VarLoopOff
\end{VCPicture}}
```

There exists<sup>7</sup> also a `\FinalStateVar` command, with the same syntax as `\StateVar` and which draws a variable state with double line, (independently of the current drawing style for the state), exactly in the same way as does `\FinalState`.

## 2.4 Very small states

There is actually a fourth size for states, called *very small* states. To define such a state one uses the command `\VSSState` with the same syntax as `\State`. However, it is not possible, of course, to put a label inside such a state.



```
\begin{VCPicture}{(-3,-3)(3,3)}
% states
\State[A]{(0,0)}{A}
\VSSState{(-2,2)}{G1} \VSSState{(-2,0)}{G2} \VSSState{(-2,-2)}{G3}
\VSSState{(2,2)}{D1} \VSSState{(2,0)}{D2} \VSSState{(2,-2)}{D3}
\EdgeL{G1}{A}{a_1} \EdgeL{G2}{A}{a_2} \EdgeL{G3}{A}{a_3}
\EdgeL{A}{D1}{b_1} \EdgeL{A}{D2}{b_2} \EdgeL{A}{D3}{b_3}
\end{VCPicture}}
```

## 2.5 Miscellaneous

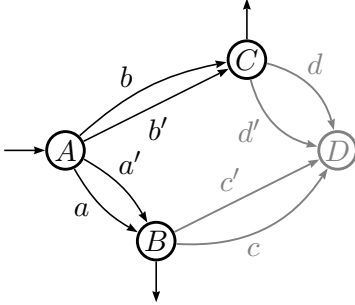
A few commands allow to tune the drawing of an automaton without going to precise mastering of the parameters involved in the commands and that we shall describe in section 4.

### Dimmed states and transitions

<code>\DimState</code>	<code>\DimEdge</code>	Draw states and transitions and their respective labels in gray.
<code>\RstState</code>	<code>\RstEdge</code>	Restore the normal style.

<sup>7</sup>Since version 0.3 only.

May be used for instance in order to indicate the non accessible part of an automaton.

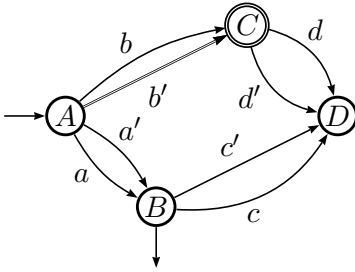


```
\begin{VCPicture}{(0,-2)(6,2)}
% states
\State[A]{(0,0)}{A} \State[B]{(2,-2)}{B} \State[C]{(4,2)}{C}
\DimState \State[D]{(6,0)}{D}
%initial-final
\Initial{A} \Final[s]{B} \Final[n]{C}
% transitions
\ArcR{A}{B}{a} \ArcL{A}{C}{b}
\DimEdge \LArcR{B}{D}{c} \LArcL{C}{D}{d} \RstEdge
\ArcL{A}{B}{a'} \EdgeR{A}{C}{b'}
\DimEdge \EdgeL{B}{D}{c'} \LArcR{C}{D}{d'}
%
\end{VCPicture}
```

### Double lines

<code>\StateLineDouble</code>	<code>\EdgeLineDouble</code>	Draw states and transitions with double lines.
<code>\StateLineSimple</code>	<code>\EdgeLineSimple</code>	Draw states and transitions with simple lines.

May be used for emphasize transitions or states of an automaton.



```
\begin{VCPicture}{(0,-2)(6,2)}
% states
\State[A]{(0,0)}{A} \State[B]{(2,-2)}{B}
\StateLineDouble \State[C]{(4,2)}{C}
\StateLineSimple \State[D]{(6,0)}{D}
%initial-final
\Initial{A} \Final[s]{B}
% transitions
\ArcR{A}{B}{a} \ArcL{A}{C}{b}
\LArcR{B}{D}{c} \LArcL{C}{D}{d}
\ArcL{A}{B}{a'} \EdgeLineDouble \EdgeR{A}{C}{b'}
\EdgeLineSimple \EdgeL{B}{D}{c'} \LArcR{C}{D}{d'}
%
\end{VCPicture}
```

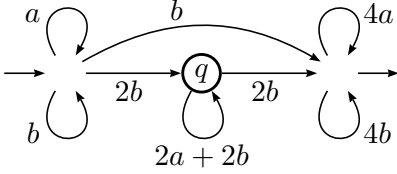
The command `\FinalState{(x,y)}{p}` is equivalent to the sequence `\StateLineDouble \State{(x,y)}{p} \StateLineSimple` if the current implicit mode is `\StateLineSimple`, it is equivalent to `\State{(x,y)}{p}` in the other case.

### Hiding states

<code>\HideState</code>	Cancel the drawing of states.
<code>\ShowState</code>	Active the drawing of states.

May be useful for drawing slides that will be overlayed.<sup>8</sup>

<sup>8</sup>We began to write the package when overhead projectors were still of common usage...



```
\begin{VCPicture}{(0,-2)(6,2)}
% states
\HideState
\State[i]{(0,0)}{A} \State[t]{(6,0)}{C}
\ShowState
\State[q]{(3,0)}{B}
% initial-final
\Initial{A} \Final{C}
% transitions
\EdgeR{A}{B}{2b} \EdgeR{B}{C}{2b} \LArcL{A}{C}{b}
\LoopN{A}{a} \LoopS{A}{b} \LoopS[.5]{B}{2a+2b}
\LoopN[.75]{C}{4a} \LoopS[.75]{C}{4b}
%
\end{VCPicture}
```

## Reverse transitions

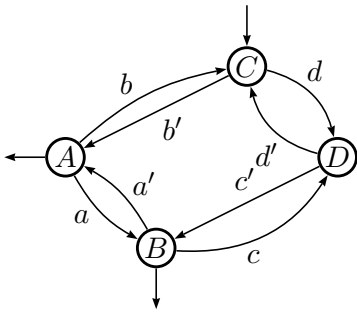
`\ReverseArrow`

Exchange the origin and the end of a transition.

`\StraightArrow`

Restore the normal style.

May be useful for modifying a part of a figure that had already been written.



```
\begin{VCPicture}{(0,-2)(6,2)}
% states
\State[A]{(0,0)}{A} \State[B]{(2,-2)}{B} \State[C]{(4,2)}{C}
\State[D]{(6,0)}{D}
% transitions
\Final[s]{B}
\ArcR{A}{B}{a} \ArcL{A}{C}{b} \LArcR{B}{D}{c} \LArcL{C}{D}{d}
\ReverseArrow
\Initial{A} \Final[n]{C}
\ArcL{A}{B}{a'} \EdgeR{A}{C}{b'} \EdgeL{B}{D}{c'}
\LArcR{C}{D}{d'}
%
\end{VCPicture}
```

## Edge border

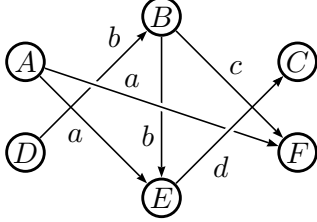
`\EdgeBorder`

Draw edges with a white border on each side.

`\EdgeBorderOff`

Restore the normal style.

May be used to make more readable a picture in which many arrows cross each other. The *order* in which the edges are drawn becomes then meaningful.



```
\begin{VCPicture}{(0,-2)(6,2)}
% states
\State[A]{(0,1)}{A} \State[B]{(3,2)}{B} \State[C]{(6,1)}{C}
\State[D]{(0,-1)}{D} \State[E]{(3,-2)}{E} \State[F]{(6,-1)}{F}
% transitions
\EdgeR[A]{E}{a} \EdgeL[.8]{D}{B}{b} \EdgeR[.7]{B}{E}{b}
%
\EdgeBorder
\EdgeL[.35]{A}{F}{a} \EdgeR[.3]{E}{C}{d}
\EdgeBorderOff
\EdgeL[B]{F}{c}
\end{VCPicture}
```

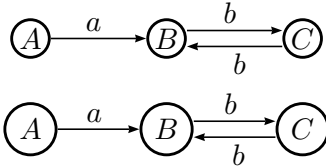
**Forth and back offset** Normally, an edge lays on the line that links the center of the start and end states. Which forbid to have two straight edges in opposite directions between states.

`\ForthBackOffset`

Shift the edge on the left handside.

`\RstEdgeOffset`

Restore the normal style.



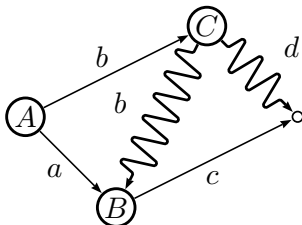
```
\begin{VCPicture}{(0,-2)(6,2)}
% states
\State[A]{(0,1)}{A} \State[B]{(3,1)}{B} \State[C]{(6,1)}{C}
\LargeState
\State[A]{(0,-1)}{A1} \State[B]{(3,-1)}{B1}
\State[C]{(6,-1)}{C1}
% transitions
\EdgeL[A]{B}{a}
\ForthBackOffset \EdgeL[B]{C}{b} \EdgeL[C]{B}{b}
\RstEdgeOffset \EdgeL[A1]{B1}{a}
\ForthBackOffset \EdgeL[B1]{C1}{b} \EdgeL[C1]{B1}{b}
\end{VCPicture}
```

## Zigzag edges

`\ZZEdgeL[Pos]{Start}{End}{Label}`

Draws a “zigzag” transition from the state *Start* to the state *End* with the label *Label* on the left or the right side.

`\ZZEdgeR[Pos]{Start}{End}{Label}`



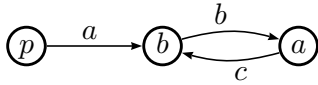
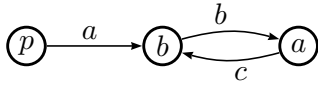
```
\begin{VCPicture}{(0,-2)(6,2)}
% states
\State[A]{(0,0)}{A} \State[B]{(2,-2)}{B} \State[C]{(4,2)}{C}
\VSState{(6,0)}{D}
% transitions
\EdgeR[A]{B}{a} \EdgeL[A]{C}{b} \EdgeR[B]{D}{c}
\ZZEdgeL[C]{D}{d} \ZZEdgeR[C]{B}{b}
%
\end{VCPicture}
```

**Lining up** By default, labels of states are centered. When states are aligned, it may happen that this seems ungainly. The following command solves this problem.

`\AlignedLabel` Put labels of states on a virtual baseline.

`\FloatingLabel` Center the label of states vertically.

Default option is `\FloatingLabel`.



```
\begin{VCPicture}{(0,-4)(6,1)}
% states
\State[p]{(0,0)}{A} \State[b]{(3,0)}{B} \State[a]{(6,0)}{C}
\AlignedLabel
\State[p]{(0,-3)}{A1} \State[b]{(3,-3)}{B1}
\State[a]{(6,-3)}{C1}
%transitions
\EdgeL{A}{B}{a} \ArcL{B}{C}{b} \ArcL{C}{B}{c}
\EdgeL{A1}{B1}{a} \ArcL{B1}{C1}{b} \ArcL{C1}{B1}{c}
%
\end{VCPicture}%
```

**The command `\Point`** This commands is a direct translation of a command from PSTricks.

`\Point{(x,y)}{Name}` Sets a point with name *Name* at the coordinate  $(x,y)$ .

This command does *not* draw any point. With PSTricks, and thus with VAUCANSON-G, any line or curve must be drawn *between* two predefined objects. That is the aim of `\Point`. For instance, initial and final arrows are drawn with this trick: any time a state is defined, eight points are silently and implicitly defined around it.

## 2.6 Writing composite labels

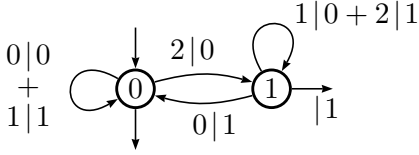
Some useful tools are proposed to write composite labels on arrows.

`\IOL{Input}{Output}` Produces the string *Input|Output*, useful when one draws transducers. This allows to modify the way it is done by redefining command `\IOL`.

`\StackXLabels{Label1}{Label2}` In these commands, *X* equals **Two** or **Three**. In the later case, commands have obviously three arguments.

`\StackXLabelsP{Label1}{Label2}` `\StackXLabels` gives a vertical arrangement of labels, whereas `\StackXLabelsP` inserts symbols + between them. `\LineXLabelsP` is the horizontal version of `\StackXLabelsP`; it can be redefined in order to change symbols, for instance.

The following automaton is the right sequential transducer that realizes the addition in base 2.



```
\begin{VCPicture}{(-2,-2)(5,3)}
% states
\State[0]{(0,0)}{A} \State[1]{(3,0)}{B}
\Initial[n]{A}
\Final[s]{A} \FinalR[e]{B}{\IOL{}{1}}
%transitions
\LoopW[.5]{A}{\StackTwoLabelsP{\IOL{0}{0}}{\IOL{1}{1}}}}
\LoopN[.7]{B}{\LineTwoLabelsP{\IOL{1}{0}}{\IOL{2}{1}}}}
\ArcL{A}{B}{\IOL{2}{0}}
\ArcL{B}{A}{\IOL{0}{1}}
%
\end{VCPicture}
```

### 3 Composing a picture

The commands described above allow to define most of the figures representing automata in any textbook or technical paper. Before turning to more sophisticated commands that are used for less than 5 percent of the elements and that appear in less than 10 percent of the figures, let us turn to few commands that help in composing the figures.

#### 3.1 Structuring the automaton: The `\VCPut` command

In many automata, a same pattern in states, and also in edges, is reproduced several times in the figure. The idea is not to try to “program” these regular patterns but to allow the use of the “copy-and-paste” operations in the source file as much as possible, with minimum corrections afterwards. This will be achieved easily with the help of the command `\VCPut`.

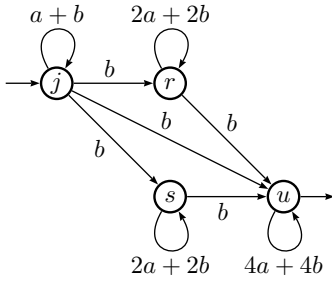
`\VCPut{(x,y)}{ instructions...}`

Translates the result of the instructions by the vector  $(x, y)$ .

Note that this command is taken directly from `PSTricks` but uses `VAUCANSON-G`’s syntax.

Of course the commands that are to be put as the second argument of `\VCPut` are the defining states commands, with possibly other `\VCPut` inside. The structuration of the figure *via* the use of `\VCPut` commands does not only save typing. It makes also easier the tuning of the figure, and the reusability of its parts.

In the example below, the two `\VCPut` are used to place the two parts of the automaton.

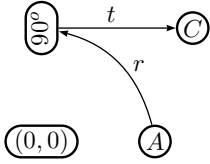


```
\begin{VCPicture}{(-1,-2)(7,5)}
\VCput{(0,3)}{\State[j]{(0,0)}{A1} \State[r]{(3,0)}{B1}}
\VCput{(3,0)}{\State[s]{(0,0)}{A2} \State[u]{(3,0)}{B2}}
%
\Initial{A1} \Final{B2}
\EdgeL{A1}{B1}{b} \LoopN[.5]{A1}{a+b} \LoopN[.5]{B1}{2a+2b}
\EdgeR{A2}{B2}{b} \LoopS[.5]{A2}{2a+2b} \LoopS[.5]{B2}{4a+4b}
\EdgeR{A1}{A2}{b} \EdgeL{B1}{B2}{b} \EdgeL{A1}{B2}{b}
%
\end{VCPicture}
```

The `\VCput` command can also be used to rotate a figure by using an optional parameter.

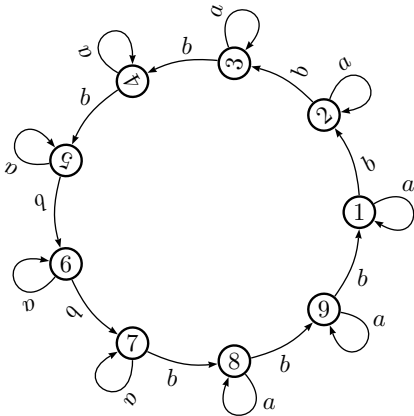
`\VCput[r]{(x,y)}{ instructions...}`

Rotate the result of the instructions with angle  $r$  and center  $(0, 0)$  and then translate it by the vector  $(x, y)$ .



```
\begin{VCPicture}{(-1,-1)(5,4)}
\StateVar[(0,0)]{(0,0)}{0}
\State[A]{(3,0)}{A}
\VCput[90]{(0,0)}{\StateVar[90^o]{(3,0)}{B}}
\VCput[90]{(4,0)}{\State[C]{(3,0)}{C}}
\LArcR{A}{B}{r} \EdgeL{B}{C}{t}
\end{VCPicture}
```

The default option is that labels of states (except those of *variable states*) are put in a horizontal position (`\RigidLabel`), but the command `\SwivelLabel` makes them rotate with the figure.



```
\begin{VCPicture}{(-5,-5)(5,5)}
\SwivelLabel
\State[1]{(4,0)}{A1}\LoopE{A1}{a}
\VCput[40]{(0,0)}{%
\State[2]{(4,0)}{A2}\LoopE{A2}{a}\ArcR{A1}{A2}{b}}
\VCput[80]{(0,0)}{%
\State[3]{(4,0)}{A3}\LoopE{A3}{a}\ArcR{A2}{A3}{b}}
\VCput[120]{(0,0)}{\State[4]{(4,0)}{A4}\LoopE{A4}{a}}
\VCput[160]{(0,0)}{\State[5]{(4,0)}{A5}\LoopE{A5}{a}}
\RigidLabel
\VCput[200]{(0,0)}{%
\State[6]{(4,0)}{A6}\LoopE{A6}{a}\ArcR{A5}{A6}{b}}
\VCput[240]{(0,0)}{%
\State[7]{(4,0)}{A7}\LoopE{A7}{a}\ArcR{A6}{A7}{b}}
\VCput[280]{(0,0)}{\State[8]{(4,0)}{A8}}
\VCput[320]{(0,0)}{\State[9]{(4,0)}{A9}}
\ArcR{A3}{A4}{b} \ArcR{A4}{A5}{b}
\ArcR{A7}{A8}{b} \ArcR{A8}{A9}{b} \ArcR{A9}{A1}{b}
\LoopL{A8}{280}{a} \LoopL{A9}{320}{a}
\end{VCPicture}
```

One can notice that `\RigidLabel` only acts on labels of states. To draw arcs with “rigid” labels, commands have to be written outside `\VCput`. We shall see later how to deal with the command `\LoopL` in order to give any orientation to loops without using `\VCput`.



### 3.2 Using separate files

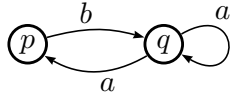
In order to use several times the same figure in the same or in different papers, or even, as we shall see in the next section, in two different contexts such as in a paper and in a slide for a talk, it is convenient to have the figure written into a file and imported in the main  $\text{\LaTeX}$  source file. This can be handled by the following macros.

<code>\VCCall{FileName}</code>	Calls the file <i>FileName</i> and apply the <code>\VCScale</code> .
<code>\SetVCDirectory{Directory}</code>	Sets to <i>Directory</i> the directory where the file <i>FileName</i> will be taken by the macro <code>\VCCall</code> . Set to the current directory by default. Usual Unix syntax is used to name directories.
<code>\ShowName</code> <code>\HideName</code>	Sets or unsets a flag that controls the printing of <i>FileName</i> besides the figure drawn by <code>\VCCall</code> .

For the example below, the following lines have been saved in the file `TwoStates.tex`:

```
\begin{VCPicture}{(-1,-1)(4,1)}
\State[p]{(0,0)}{A} \State[q]{(3,0)}{B}%
\ArcL{A}{B}{b} \LArcL{B}{A}{a} \LoopE{B}{a}
\end{VCPicture}
```

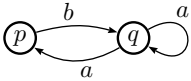
`\VCCall` applies the same parameters to figures described in files as `\VCDraw` does.



`\VCCall{TwoStates}`

`\SmallPicture \ShowName`  
`\VCCall{TwoStates}`

TwoStates

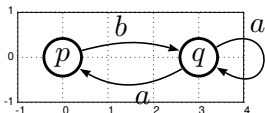


### 3.3 Frame and grid

Two macros that help placing the figure in the page, and the states in the figure.

<code>\ShowFrame</code> <code>\HideFrame</code>	Sets or unsets a flag that controls the drawing of the <i>bounding box</i> which is defined by the parameter of the <code>\VCPicture</code> environment.
<code>\ShowGrid</code> <code>\HideGrid</code>	Sets or unsets a flag that controls the drawing of a <i>grid</i> inside the bounding box which is defined by the parameter of the <code>\VCPicture</code> environment.

Default settings are `\HideFrame` and `\HideGrid`.

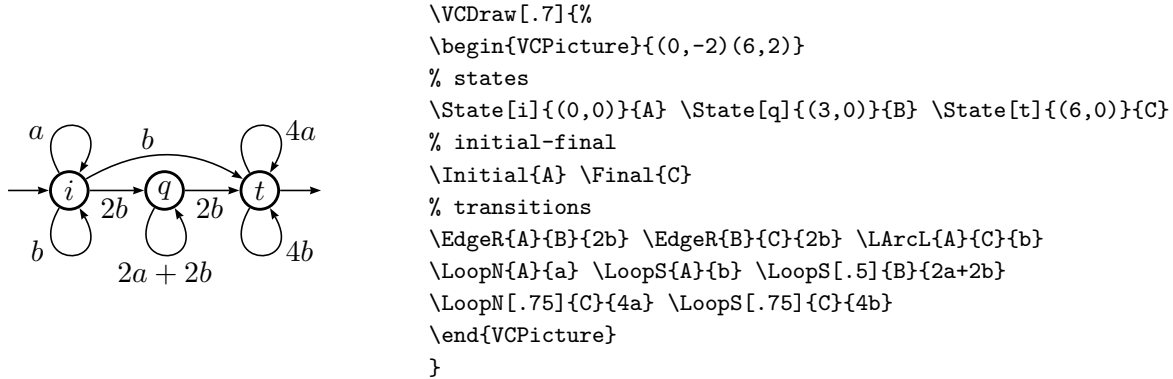


`\ShowFrame \ShowGrid \VCCall{TwoStates}`

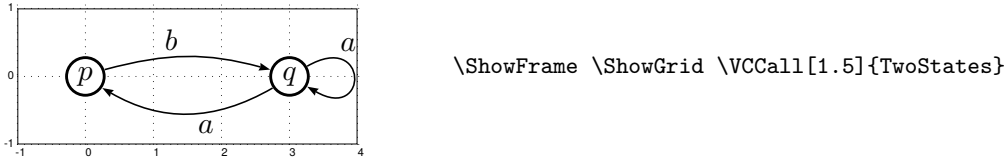
### 3.4 The grid scale

This parameter controls the value of the `PSTricks` unit length. As values that define the size of states and the width of lines are given with fixed units (as `cm` or `pt`), they are not modified by this parameter. However, the scale of the grid on which they are put, *i.e.* the distance between each other is modified by grid scale. This parameter is actually the first (optional) argument of `\VCDraw` or `\VCCall`.

The following example is the figure of Subsection 1.6 with another grid scale:



We can call `TwoStates` with another grid scale.



## 4 Parametrized drawing commands

In the sequel, the letter `X` stands for either `L` or `R`.

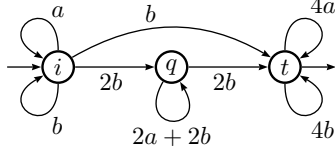
### 4.1 Parametrized loops

One can draw loops around state with any direction.

`\LoopX[Pos]{Direction}{Name}{Label}`  
`\CLoopX[Pos]{Direction}{Name}{Label}`

Draw a loop around the state *Name* in direction *Direction*, which is an angle with respect to east, with label *Label*.

As the label is placed outside the loop, `\LoopL` is reversed with respect to `\LoopR`. The opening of the loop is  $60^\circ$  for the `LoopX` commands,  $44^\circ$  for the `CLoopX` commands.



```
\begin{VCPicture}{(0,-2)(6,2)}
% states
\State[i]{(0,0)}{A} \State[q]{(3,0)}{B} \State[t]{(6,0)}{C}
% initial-final
\Initial{A} \Final{C}
% transitions
\EdgeR{A}{B}{2b} \EdgeR{B}{C}{2b} \LArcL{A}{C}{b}
\LoopR{120}{A}{a} \LoopL{-120}{A}{b} \LoopS[.5]{B}{2a+2b}
\LoopL[.5]{60}{C}{4a} \LoopR[.5]{-60}{C}{4b}
%
\end{VCPicture}%
```

## 4.2 Parametrized arcs

`\VArcX[Pos]{arcangle=x,ncurv=v}{Start}{End}{Label}`

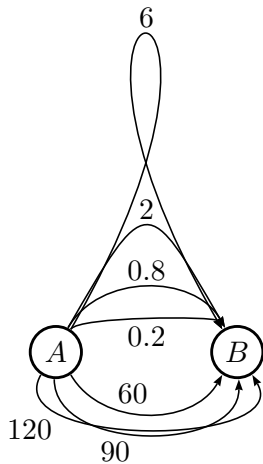
Draw an arc from *Start* to *End* with label *Label*, with a starting angle *x* and an eccentricity *v*.

The starting angle is defined by the parameter `arcangle=x` and is measured, going to the left if `X=L`, to the right if `X=R`, from the line that goes from *Start* to *End*, called the *base line*. The parameter `ncurv` is the ‘eccentricity’ of the arc and some how measures the distance between the center of the arc and the base line as one can observe on the figure below.

The first (non optional) parameter of `\VArcX` contains PSTrick’s parameters, written with PSTrick’s syntax: they are separated by a comma and there must be no space inside the braces that contains them. All these parameters are optional, but this first parameter of `\VArcX` itself is not an optional argument, which means that this command must always have four arguments (and possibly an optional one):

`\VArcL{arcangle=15}{A}{B}{x}`  
`\VArcL{ncurve=1}{A}{B}{x}` are syntactically correct commands.  
`\VArcL{}{A}{B}{x}`

Default value for `arcangle` is the same as for `\LArc`, whereas the default `ncurv` value is 1.



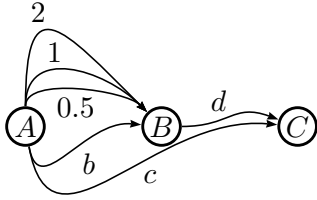
```
\begin{VCPicture}{(-3,-2)(3,6)}
% states
\LargeState
\State[A]{(-2,0)}{A}
\State[B]{(2,0)}{B}
%transitions
\VArcR[.5]{arcangle=45,ncurv=.2}{A}{B}{0.2}
\VArcL[.5]{arcangle=45,ncurv=.8}{A}{B}{0.8}
\VArcL[.5]{arcangle=45,ncurv=2}{A}{B}{2}
\VArcR[.5]{arcangle=45,ncurv=6}{A}{B}{6}
\VArcL{arcangle=-60}{A}{B}{60}
\VArcR{arcangle=-90}{A}{B}{90}
\VArcR[.2]{arcangle=-120}{A}{B}{120}
%
\end{VCPicture}
```

### 4.3 General curve

Draw a curve from *Start* to *End* with label *Label*, starting with an angle  $x$ , arriving with an angle  $y$  and with an eccentricity  $v$ .

`\VCurveX[Pos]{angleA=x,angleB=y,ncurv=v}{Start}{End}{Label}`

A (Bézier) curve, computed by PSTricks with the three parameters: the *starting angle* fixed by `angleA=x`, the ending angle fixed by `angleB=y`, and the eccentricity `ncurv=v`. The two angles are measured, as trigonometric angles are, from the East direction and counterclockwise. The syntax of these parameters is PSTricks's and they are optional, as in `\VArcX`. Default values for `angleA` and `angleB` are respectively 0 and 180; `ncurv` is fixed by default to 1.



```
\begin{VCPicture}{(-4,-4)(4,4)}
%states
\State[A]{(-3,0)}{A}
\State[B]{(0,0)}{B}
\State[C]{(3,0)}{C}
%transitions
\VCurveL{angleA=90,angleB=135}{A}{B}{1}
\VCurveR[.5]{angleA=90,angleB=135,ncurv=.5}{A}{B}{0.5}
\VCurveL{angleA=90,angleB=135,ncurv=2}{A}{B}{2}
\VCurveR[.6]{angleA=-80,angleB=175}{A}{B}{b}
\VCurveR[.6]{angleA=-80,angleB=175}{A}{C}{c}
\VCurveL{angleB=20,angleB=160}{B}{C}{d}
\end{VCPicture}
```

### 4.4 Transitions labelling

**Initial and final arrows with a label** When one deals with transducers, or automata with multiplicity, one may need to put some labels on initial or final arrows.

`\InitialL[Pos]{Dir}{Name}{Label}`

Makes the state *Name* initial and writes *Label* on the left or the right size of the incoming arrow.

`\InitialR[Pos]{Dir}{Name}{Label}`

`\FinalL[Pos]{Dir}{Name}{Label}`

Makes the state *Name* final and writes *Label* on the left or the right size of the outgoing arrow.

`\FinalR[Pos]{Dir}{Name}{Label}`

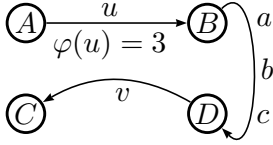
Here, the parameter *Dir* is not optional. Whereas, there is an optional parameter *Pos* that defines the position of the label on the arrow. Its default values is fixed to 0.1 (*resp.* 0.9) for initial (*resp.* final) arrows. The label is thus written far enough of the state.

**Independant labelling of transitions** There exists commands to label transitions that have not been labeled yet or to put a second label on them. These commands must directly follow the definition of the transition that will be labeled.

`\LabelL[Pos]{Label}`

`\LabelR[Pos]{Label}`

Puts a label on the previous transition. The default position of the label is the same as for edges.



```
\begin{VCPicture}{(-3,-2)(3,2)}
\State[A]{(-2,1)}{A} \State[B]{(2,1)}{B}
\State[C]{(-2,-1)}{C} \State[D]{(2,-1)}{D}
%
\EdgeL{A}{B}{u} \LabelR{\varphi(u)=3}
\LArcR{D}{C}{} \LabelL{v}
\VCurveL[.3]{angleA=45,angleB=-45}{B}{D}{a}
\LabelL[.5]{b} \LabelL[.7]{c}
\end{VCPicture}
```

## 4.5 Miscellaneous

**Loops on variable states** Instead of using `\VarLoopN` or `\VarLoopS`, one can modify parameters of loops to draw, for instance, many loops on variable states. As usual, these loops may be drawn on the south or north side of the states only.

`\VarLoopOn`

Modifies parameters in order to draw loops around variable states.

`\VarLoopOff`

Restore the normal style of loops.

Command `\LoopVarN` is equivalent to the sequence `\LoopVarOn \LoopN \LoopVarOff`.

**Saving memory** In default mode, the call to a `\State` command defines eight additional points, that allow then to draw initial and final arrows in the prescribed compass direction. In case of large figures, with many states, this ought to lead to an overflow of  $\text{\TeX}$  memory. This is the reason why one can define states without any additional point.

`\PlainState`

Draw states with no additional points.

`\FullState`

Draw states with eight points to draw initial or final arrows.

Of course, when in `\PlainState` mode, one can define states with additional points.

`\IFState[Label]{(x,y)}{Name}`

Draws a state with four points  $(n, s, e, w)$ .

`\IFXState[Label]{(x,y)}{Name}`

Draws a state with eight points  $(n, s, e, w, ne, se, nw, sw)$ .

In `\PlainState` mode, one should not apply an `\Initial` or a `\Final` command to a state defined by `\State`. Likewise, these commands can be only used for `\IFState` with directions  $n, s, e$  and  $w$ .

Note that even when in `\FullState` mode, variable states defined by `\StateVar` have the same four additional points as `\IFState`.

## 5 Modifying and setting the parameters

The reason why the commands in VAUCANSON-G are simple and concise is that all the parameters that control the geometry (size, angles, ...) and the drawing (line width, line style, line color, ...) of the states and transitions in an automaton are preset by VAUCANSON-G and can be kept implicit. But there are two instances where a user may want to modify these preset values.

The first case is when one needs to tune a parameter in order to make the figure more readable, or nicer. Such a case has already been considered in sections 1.3, 1.4 and 1.5, where the parameter *Pos* that controls the position of the label of a transition along the edge has been modified with an optional argument. The parameters that will be considered in this section are not optional parameters of already defined commands but parameters that are set with their own commands.

The other case where a user will change the implicit parameters of a figure or, more likely, of a set of figures, is when he (or she) will be in the position to change completely the “style” of the drawing. For instance, a same set of figures may be used in a paper *and* in a series of slides for the presentation of the paper. Of course, the figures have to be set at a larger scale, probably in colour, and possibly with different relative sizes of the elements.

We begin with the description of the options for the latter case, which is by far simpler. We then explain the syntax convention taken to coin the names of the many commands that control the many parameters. Then comes the list of these parameters, for scaling first, then for states, for transitions, and finally for the fine tuning of some details of the drawing.

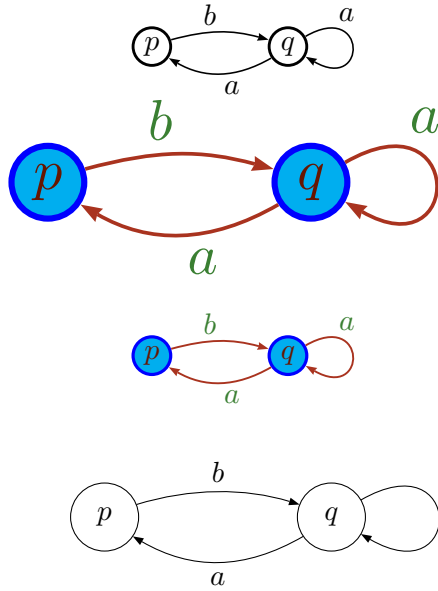
### 5.1 Style files

The package VAUCANSON-G is loaded with default values that are all set up in the file `VCPref-default.tex`. The package comes with two other styles, in particular one for drawing slides. The normal way of using the different style files is to call the wrapper `vaucanson-g` with an option:

```
\usepackage[option]{vaucanson-g}
```

The default option is **default**; the other possible options are **slides**, **beamer** and **mystyle**.

Although it is not the normal usage, it is also possible to change style in the course of the text by loading a style file with an `\input` command. In this case, it is recommended to reset all parameters by calling `VCPref-default.tex`.



```

\usepackage{vaucanson-g}
...
\VCCTwoStates
...

\usepackage[slides]{vaucanson-g}
...
\VCCTwoStates
...

\usepackage[beamer]{vaucanson-g}
...
\VCCTwoStates
...

...
\input{VCPref-default} % gets back to default values
\input{VCPref-mystyle}
\VCCTwoStates

```

By editing the files `VCPref-mystyle`, `VCPref-beamer` or `VCPref-slides`, and giving the parameters the values he (or she) thinks more appropriate, a user may easily build his (or her) own style for drawing automata. Modifying directly `VCPref-default`, although possible, is not necessarily a good idea. By cut, editing and paste, it is not difficult create many style files, and even, with an easy modification of the wrapper itself to make them possible options. The content of these style files are described in the following subsections. `VCPref-slides`, `VCPref-beamer` and `VCPref-mystyle` are given at Appendix B.

## 5.2 Syntax of parameter settings

The set of values of all parameters defines a *style* of figures. Most of parameters, *e.g.* `\StateLineColor` which defines the color of the line that delimits a state, may be given a *temporary* value, within a figure, without changing the style. These parameters are dealt with by means of 3 different commands:

<code>\ChgParam{ Val }</code>	Gives the new value <i>Val</i> to <code>\Param</code> .
<code>\RstParam</code>	Restores the style value in <code>\Param</code> .
<code>\SetParam{ Val }</code>	Sets the style value for <code>\Param</code> to <i>Val</i> .

The command `\SetParam` can be called *outside* a `\VCPicture` environment, in which case it will be effective in all subsequent figures, until a new setting of the same parameter. Its natural place, if it is considered as a modification of the style, is in the preamble, after the call to the package  $\overline{\text{VAUCANSON-G}}$ . If the command `\SetParam` is called *inside* a `\VCPicture` environment, its effect will be restricted to that figure.

Such parameters will be called *adjustable parameters* in the sequel.

Note that in the case where `\Param` is a value (a *length* or a *scale*), the value of `Val` in `\ChgParam{Val}` is a *coefficient* which is applied to the style value of `\Param` defined with `\SetParam{Val}`. Therefore, in this case, `\RstParam` is equivalent to `\ChgParam{1}`.

The other parameters, as those that define the “dimmed” style, are called *preset parameters*.

There is finally a class of parameters that fall in between the adjustable and the preset ones, those for which several values are “preset” but that can be given other values. These parameters will be called *multivalued preset parameters*.

### 5.3 Scale parameter (Multivalued preset parameter)

`\FixVCScale{Scale}` Sets `\VCScale` to *Scale*.

The commands `\MediumPicture`, `\SmallPicture`, `\TinyPicture` and `\LargePicture` set `\VCScale` to the values `\MediumScale`, `\SmallScale`, `\TinyScale` and `\LargeScale` respectively. Their values are

<code>\TinyScale</code>	<code>\SmallScale</code>	<code>\MediumScale</code>	<code>\LargeScale</code>
0.42	0.5	0.6	0.85

These preset values can be modified by a `\renewcommand`.

### 5.4 State parameters

**State diameter** (Multivalued preset parameter) This parameter controls, *via* the commands `\MediumState`, `\SmallState` or `\LargeState` the value of a number of other preset parameters.

`\FixStateDiameter{Diam}` Sets the state diameter to *Diam* (*Diam* is a length) until the next call to a `\ZState` command, or to the next figure if called inside a `\VCPicture`.

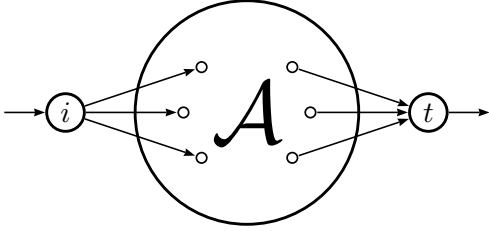
Preset diameter sizes are:

<code>\VerySmallState</code>	<code>\SmallState</code>	<code>\MediumState</code>	<code>\LargeState</code>
0.3cm	0.6cm	0.9cm	1.2cm

These values can be modified by a `\renewcommand`. However, as commands that call these parameters deal also with other parameters as the eccentricity of loops or the size of initial and final arrows, changing these values without modifying the other ones can lead to an ungainly result.

The shape of a normalized automaton can be drawn in the following way:





```
\begin{VCPicture}{(-5,-3)(5,3)}
\FixStateDiameter{5cm} \ChgStateLabelScale{3}
\State[{\cal A}]{(0,0)}{A}
\MediumState \RstStateLabelScale
\State[i]{(-4,0)}{I} \State[t]{(4,0)}{T}
\VSState{(-1,-1)}{P1} \VSState{(1,-1)}{Q1}
\VSState{(-1,1)}{P2} \VSState{(1,1)}{Q2 }
\VSState{(-1.4,0)}{P3} \VSState{(1.4,0)}{Q3}
\Initial{I} \Final{T}
%
\EdgeL{I}{P1}{ } \EdgeL{I}{P2}{ } \EdgeL{I}{P3}{ }
\EdgeL{Q1}{T}{ } \EdgeL{Q2}{T}{ } \EdgeL{Q3}{T}{ }
\end{VCPicture}
```

**State adjustable parameters** Each following parameter can be modified by the command `\ChgParameter`. The style value can be restored by `\RstParameter` and one can set the style value with `\SetParameter`.

Parameter	Default value	Possible values
<i>StateLineStyle</i>	solid	dashed, dotted, none
<i>StateLineWidth</i>	1.8pt	
<i>StateLineColor</i>	black	red, lightgray, ...
<i>StateLabelColor</i>	black	blue, Salmon, ...
<i>StateLabelScale</i>	1.7	
<i>StateFillStatus</i>	solid	vlines, hlines, crosshatch, none
<i>StateFillColor</i>	white	green, Purple, ...

As it has been previously noticed, the argument of `\ChgStateLineWidth` is not a **length** but a coefficient applied to the style width. For instance, with the default value of *StateLineWidth*, after `\ChgStateLineWidth{2}` edges will be drawn with a 3.6pt width.

When pictures are drawn to scale `\VCScale` that is set to 0.6, printing state labels with scale 1.7 makes them to be as large as characters of the text. As for line width, `\ChgStateLabelScale{.5}` sets the label scale to 0.85 that make them twice smaller than characters of the text (because of `\VCScale`).

**State preset parameters** They deal with the definition of “dimmed” states.

Parameter	Default value
<i>DimStateLineStyle</i>	<b>solid</b>
<i>DimStateLineColor</i>	<b>gray</b>
<i>DimStateLineCoef</i>	<b>1</b>
<i>DimStateLabelColor</i>	<b>gray</b>
<i>DimStateFillColor</i>	<b>white</b>

*DimStateLineCoef* is a coefficient that is applied to *StateLineWidth* in drawing dimmed states. It might be larger than 1 to compensate optical effects when *DimStateLineStyle* is **dashed** or **dotted**.

The style of dimmed states can be fixed by the command `\FixDimState{LineStyle}{LineColor}{LineCoef}{LabelColor}{FillColor}`.

In mode *StateLineDouble* or when one uses the command `\FinalState`, two parameters give the width of each of both lines and the distance between them. These values are coefficients that are applied to *StateLineWidth*.

Parameter	Default value
<i>StateLineDblCoef</i>	<b>0.6</b>
<i>StateLineDblSep</i>	<b>0.4</b>

These values can be fixed by the command `\FixStateLineDouble{Coef}{Sep}`.

## 5.5 Transition parameters

**Edge adjustable parameters** Each following parameter can be modified by the command `\ChgParameter`. The style value can be restored by `\RstParameter` and one can set the style value with `\SetParameter`.

Parameter	Default value	Possible values
<i>EdgeLineStyle</i>	<b>solid</b>	<b>dashed, dotted, none</b>
<i>EdgeLineWidth</i>	<b>1pt</b>	
<i>EdgeLineColor</i>	<b>black</b>	many colours
<i>EdgeLineDblStatus</i>	<b>false</b>	<b>true</b>
<i>EdgeNodeSep</i>	<b>0pt</b>	
<i>EdgeLabelColor</i>	<b>black</b>	many colours
<i>EdgeLabelScale</i>	<b>1.7</b>	
<i>EdgeLabelSep</i>	<b>3.5pt</b>	

As for states, the argument of `\ChgEdgeLineWidth` is a coefficient applied to the style line width.

The parameter *EdgeNodeSep* defines a distance between the begining, and the end, of a transition and the state it starts from and arrives to. The default value is `0pt`, that is, no distance at all; but it may be the case, especially when there is no line around the state that one may want to increase the distance between the transition and the label of the state.

The parameter *EdgeLabelSep* controls the distance between the box that contains the label of the transition (built by `LATEX`) and the transition.

**Edge preset parameters** A first class of preset parameters deals with the definition of “dimmed” edges.

Parameter	Default value
<i>DimEdgeLineStyle</i>	<b>solid</b>
<i>DimEdgeLineColor</i>	<b>gray</b>
<i>DimEdgeLineCoef</i>	<b>1.2</b>
<i>DimEdgeLabelColor</i>	<b>gray</b>

The style of dimmed edges can be fixed by the command `\FixDimEdge{LineStyle}{LineColor}{LineCoef}{LabelColor}`.

The style of the border of edges (in mode `\EdgeBorder`), is defined by two parameters :

Parameter	Default value
<i>EdgeLineBorderCoef</i>	<b>2</b>
<i>EdgeLineBorderColor</i>	<b>white</b>

`\EdgeLineBorderCoef` is a coefficient applied to `\EdgeLineWidth`, that gives the width of the border.

The style of borders of edges can be fixed by the command `\FixEdgeBorder{Coef}{Color}`.

The width of the double line of (in mode *EdgeLineDouble*), is defined by two parameters, that give the width of each of both lines and the distance between them. These values are coefficients that are applied to *EdgeLineWidth*.

Parameter	Default value
<i>EdgeLineDblCoef</i>	<b>0.5</b>
<i>EdgeLineDblSep</i>	<b>0.6</b>

These values can be fixed by the command `\FixEdgeLineDouble{Coef}{Sep}`.

## 5.6 Advanced parameters

These parameters can be changed by using the usual command `\renewcommand` (or `\setlength`, when the parameter is a length).

As it has already be mentionned, the preset commands that set the diameter of states control parameters for drawing edges (the length of initial and final arrows or the curvature of loops). Their values are:

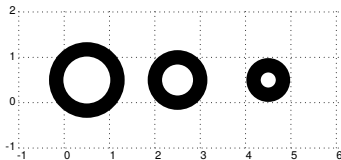
$X$	<code>\ArrowOnX</code>	<code>\LoopOnX</code>	<code>\CLoopOnX</code>
<code>LargeState</code>	1.3	5.8	6
<code>MediumState</code>	1.5	5.8	8
<code>SmallState</code>	1.7	5.8	12
<code>VariableState</code>		5.1	5.2
<code>VerySmallState</code>	5	15	

`ArrowOnX` is the coefficient that is applied to the radius of the state to obtain the length of initial or final arrows. `LoopOnX` and `CLoopOnX` is the eccentricity of loops and closed loops respectively.

Another parameter determines whether the dimension of states refer to the middle, the inner or the outer side of the boundary. There exist one parameter for classical states and one for states with double line:

Parameter	Default value
<code>\StateDimen</code>	outer
<code>\StateDblDimen</code>	middle

The possible values are `inner`, `outer` or `middle`.



```

\ShowGrid
\VCDraw{%
\begin{VCPicture}{(-1,-1)(6,2)}
\ChgStateLineWidth{5}
\FixStateDiameter{1cm}
\renewcommand{\StateDimen}{inner}
\State{(0.5,0.5)}{I}
\renewcommand{\StateDimen}{middle}
\State{(2.5,0.5)}{M}
\renewcommand{\StateDimen}{outer}
\State{(4.5,0.5)}{O}
\end{VCPicture}}
\HideGrid

```

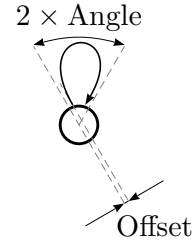
**Transition adjustable geometric parameters** All these parameters are adjustable. That means there exist commands as `\SetParameter`, `\ChgParameter` and `\RstParameter`.

Arcs between states are characterized by some adjustable parameters:

$X$	$XAngle$	$XCurvature$	$XOffset$
$Arc$	15	0.8	1pt
$LArc$	30	0.8	$=ArcOffset$

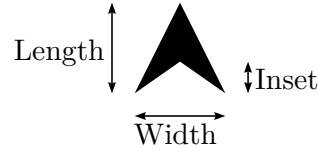
Except the curvature that depends on the size of states, parameters that define loop, the degree of opening and the offset, are adjustable parameters.

$X$	$XAngle$ (degrees)	$XOffset$
$Loop$	30	0pt
$CLoop$	22	$=LoopOffset$
$LoopVar$	28	0.7pt



Likewise, one can modify the style of arrows, that are defined by three adjustable parameters :

Parameter	Default value
$EdgeArrowStyle$	$\rightarrow$
$EdgeArrowWidth$	5pt
$EdgeArrowLengthCoef$	1.4
$EdgeArrowInsetCoef$	0.1



The *length* and the *inset* of arrows are given by coefficients that are applied to the *width*. There exists some other (non adjustable) parameters to define arrows in special cases:  $\backslash EdgeArrowRevStyle$  is equal to  $\leftarrow$ , to draw transition in reverse mode; for edges with double lines, the size of arrows is a little bit more large. It is defined by  $\backslash EdgeDblArrowWidth$  and  $\backslash EdgeDblArrowLengthCoef$  that are respectively equal to 5.5pt and 1.7.

Zigzag edges are characterized by some adjustable parameters:

Parameter	Default value
$ZZSize$	0.9cm
$ZZLineWidth$	1.7

$ZZSize$  is the apparent diameter of the zigzag and  $ZZLineWidth$  is a coefficient that is applied to  $EdgeLineWidth$  and gives the width of the line.

**Transition preset parameters** The default position of labels on transitions depends on the type of transition. For instance, on edges, this position is given by the variable `\EdgeLabelPosit` that is equal to 0.45. The first column of the following array gives these values for every transition.

X	\XLabelPosit	\XLabelRevPosit
Edge	.45	.55
Arc	.4	.6
LArc	.4	.6
Loop	.25	.75
CLoop	.25	.75
Init	.1	.9
Final	.9	.1

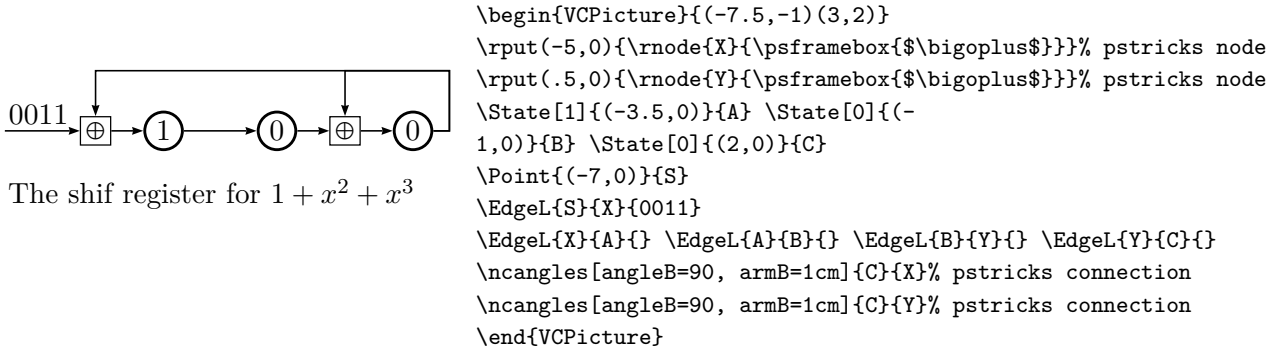
When the reverse mode is on, the position of label is given by variables that correspond to the second column, as `\EdgeLabelRevPosit` for instance. For the same type of transition, the sum of both variables that defined position for straight and reverse arrows should be equal to 1 (if one wants to keep the same appearance when using the `ReverseArrow` command).

These parameters can be modified with a `\renewcommand`.

## 6 Using “plain” PSTricks commands

VAUCANSON-G is a set of macros based on PSTricks. It is therefore fully compatible with PSTricks instructions. Actually, the `VCpicture` environment is a specialization of the `pspicture` environment; more, every state in VAUCANSON-G is a PSTricks node and every transition (loop, edge, arc) is a node connection.

The following example shows how VAUCANSON-G commands can be mixed with plain PSTricks commands. For further explanation on PSTricks, we refer to [1, 3, 4, 6].



## APPENDIX

### A Version history

A first version of the package, which would be numbered 0.1, was called *Vaucanson* and used by the authors in their papers between 2000 and 2003. But, at the same time, the authors were involved, in cooperation with the LRDE at EPITA, in the design of a C++ platform for programming automata, for which it was decided that there would be no better name than Vaucanson as well. We thus turned the name of the package for drawing automata to VAUCANSON-G.

Version 0.2 was issued in May 2003 and made available on the web together with this user's manual first version. A series of bug corrections and slight improvements made us switch at some point to version 0.3.

Just before sending the package to CTAN, we undertook a full review of the code and updated the manual accordingly. We call it version 0.4. It is the last release before a completely new version of the package, with many more options, which is currently under work.

## B Style files

### B.1 slides style

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%% Package `Vaucanson-G' version 0.4
%%
%% This is file `VCPref-slides'.
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Scales --- slides settings
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
\renewcommand{\LargeScale}{1.5}
\renewcommand{\MediumScale}{1.16}
\renewcommand{\SmallScale}{0.92}
\renewcommand{\TinyScale}{0.75}
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% State aspect
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
\SetStateLineColor{blue}
\SetStateLineWidth{2pt}
\SetStateFillColor{Cyan}
\SetStateLabelColor{Sepia}
\FixDimState{solid}{YellowOrange}{1}{YellowOrange}{white}
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Edge aspect
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
\SetEdgeLineColor{Mahogany}
\SetEdgeLineWidth{1.2pt}
\SetEdgeLabelColor{OliveGreen}
%% Dimmed edges
\FixDimEdge{dashed}{1.2}{Apricot}{Apricot}
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```



## B.2 beamer style

```

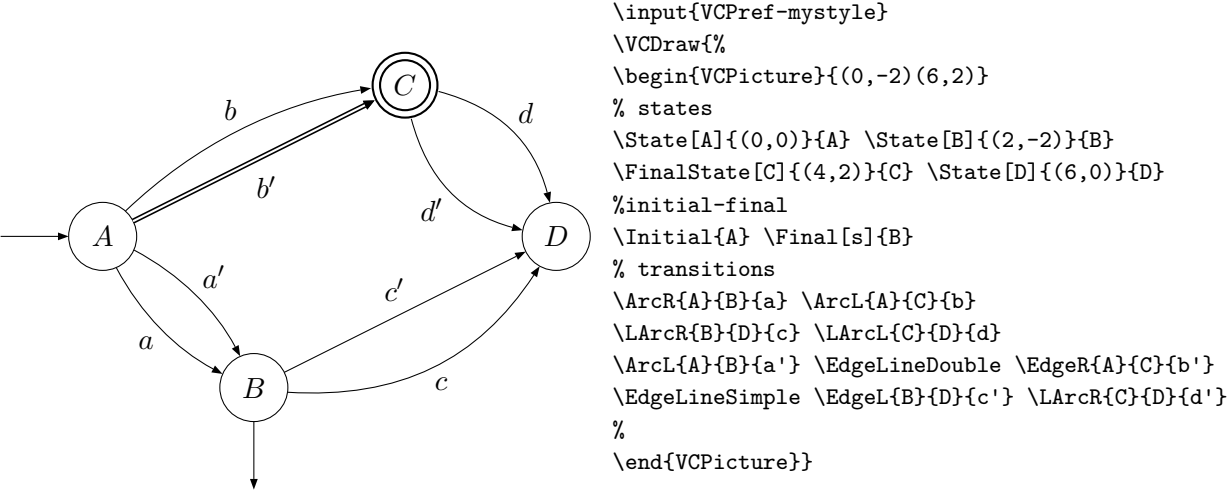
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%% Package `Vaucanson-G' version 0.4
%%
%% This is file `VCPref-slides'.
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Scales --- slides settings
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
\renewcommand{\LargeScale}{0.85}
\renewcommand{\MediumScale}{0.6}
\renewcommand{\SmallScale}{0.4}
\renewcommand{\TinyScale}{0.30}
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% State aspect
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
\SetStateLineColor{blue}
\SetStateLineWidth{2pt}
\SetStateFillColor{Cyan}
\SetStateLabelColor{Sepia}
\FixDimState{solid}{YellowOrange}{1}{YellowOrange}{white}
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Edge aspect
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
\SetEdgeLineColor{Mahogany}
\SetEdgeLineWidth{1.2pt}
\SetEdgeLabelColor{OliveGreen}
%%% Dimmed edges
\FixDimEdge{dashed}{1.2}{Apricot}{Apricot}
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

### B.3 mystyle style

This file is given only on the purpose that it can be edited by the user and called by the option `mystyle` of the wrapper (and this is the reason why the *name* `VCPref-mystyle.tex` of this file should not be changed).

The actual values given in this file as an example provide a style which mimics the way automata are drawn with GasTeX [2] as can be witnessed below.



```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%% Package `Vaucanson-G' version 0.4
%%
%% This is file `VCPref-mystyle'.
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Scales ---
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
\renewcommand{\LargeScale}{1.2} %float : argument of a \scalebox
\renewcommand{\MediumScale}{1} %float
\renewcommand{\SmallScale}{.7} %float
\renewcommand{\TinyScale}{0.5} %float
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% State parameters --- Default settings
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
\setlength{\LargeStateDiameter}{1.2cm} %length
\setlength{\MediumStateDiameter}{.8cm} %length
\setlength{\SmallStateDiameter}{.6cm} %length
\renewcommand{\StateDblDimen}{outer}
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% State aspect
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
\SetStateLineWidth{.14mm} % length
\SetStateFillStatus{none} % aspect

```

```

\SetStateFillColor{black} %% color
\SetStateLabelScale{1} %% float
\FixStateLineDouble{2}{5} %% Double style:
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Edge aspect
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
\SetEdgeLineWidth{.14mm} %% length
\SetEdgeLabelColor{black} %% color
\SetEdgeLabelScale{1} %% float
\FixEdgeLineDouble{1.5}{2} %% float :
%%% arrows
\SetEdgeArrowWidth{1.03mm} %width of the edge arrow
\SetEdgeArrowLengthCoef{1.37} %float :
\setlength{\EdgeDblArrowWidth}{1.3mm} %width :
\renewcommand{\EdgeDblArrowLengthCoef}{1.09} %
\SetEdgeArrowInsetCoef{0} %float : coef*\EdgeArrowSizeDim
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Arc geometry
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
\SetArcAngle{17} %% int (degree)
\SetLArcAngle{30} %% int (degree)
\SetArcCurvature{0.7} %% float
\SetArcOffset{1pt} %% length
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Loop geometry
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
\renewcommand{\LoopOnLargeState}{5.5} %float
\renewcommand{\LoopOnMediumState}{7} %float : curvature
\renewcommand{\LoopOnSmallState}{9} %float
\renewcommand{\LoopOnVariableState}{4.5} %float
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Edge labels positionning
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
\renewcommand{\EdgeLabelPosit}{.5} %per cent
\renewcommand{\EdgeLabelRevPosit}{.5}
\renewcommand{\ArcLabelPosit}{.5}
\renewcommand{\ArcLabelRevPosit}{.5}
\renewcommand{\LArcLabelPosit}{.5}
\renewcommand{\LArcLabelRevPosit}{.5}
\renewcommand{\LoopLabelPosit}{.5}
\renewcommand{\LoopLabelRevPosit}{.5}
\renewcommand{\CLoopLabelPosit}{.5}
\renewcommand{\CLoopLabelRevPosit}{.5}
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Initial states parameters
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
\renewcommand{\ArrowOnMediumState}{1} %float
\renewcommand{\ArrowOnSmallState}{1} %float
\renewcommand{\ArrowOnLargeState}{1} %float
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

## References

- [1] D. GIROU Présentation de PSTricks, *Cahier GUTenberg* **16** (1994) 21-70.
- [2] P. GASTIN *GasTeX: Graphs and Automata Simplified in TeX*.  
<http://www.lsv.ens-cachan.fr/~gastin/gastex/gastex.html>
- [3] M. GOOSSENS, S. RAHTZ AND F. MITTELBACH *The LaTeX Graphics Companion*. Addison Wesley, 1997.
- [4] M. GOOSSENS, S. RAHTZ, F. MITTELBACH, D. ROEGEL, AND H. VOSS *The LaTeX Graphics Companion, 2nd Edition*. Addison Wesley, 2007.
- [5] J. SAKAROVITCH, *Eléments de théorie des automates*. Vuibert, 2003. Eng. trans. *Elements of Automata Theory*. Cambridge University Press, to appear.
- [6] H. VOSS, *PSTricks – Grafik für TeX und LaTeX*. Lehmanns/Dante e.V., 4th ed. 2007.

Some examples of articles or slides that contain automata drawn with VAUCANSON-G can be found at the following URLs:

- <http://www.enst.fr/~jsaka/>
- <http://igm.univ-mlv.fr/~lombardy/>