Graphviz and Ti*k*Z

Claudio Fiandrino

December 4, 2011

Index

Introduction

- Graphviz is a tool that allows to build graphs
- In this presentation we analyse how is it possible export a Graphviz result in a Ti*k*Z picture thanks to:
  - *dot2tex*
- The operating system used is Ubuntu 11.04
- Each example contains:
  - the dot code
  - the graphical result obtained by running pdfLaTeX
- Examples are mainly derived from the web site of dot2tex

Outline

Requirements for dot2tex

As reported in the dot2tex main page, the following software are required:

- python, version 2.4 of later
- pyparsing version 1.4.8 (recommended) or later
- Graphviz
- Preview LaTeX package
  - to not have problems I suggest to install TeX Live: this helps if you want to exploit the *dot2texi* package and, of course, with Ti*k*Z/PGF
- Ti*k*Z/PGF
  - in case you have TeX Live you do not need to install anymore

Installation of pyparsing

An easy way to install things is to use easy_install.

Easy_Install

Open your terminal and do:

```
sudo apt-get install python-setuptools
```

Pyparsing

Digit, again in the terminal:

```
sudo easy_install pyparsing
```

Installation of pyparsing

An easy way to install things is to use easy_install.

Easy_Install

Open your terminal and do:

```
sudo apt-get install python-setuptools
```

Pyparsing

Digit, again in the terminal:

```
sudo easy_install pyparsing
```

Installation of Graphviz

Open Synaptic, the packet manager, and install the followings:

- graphviz, graphviz-doc, graphviz-dev, libgraphviz-dev, libgraphviz-perl
- libgv-lua, libgv-perl, libgv-php5, libgv-guile, libgv-python, libgv-ruby, libgv-tlc, libgv-ocaml

Installation of dot2tex

There are two ways to install it:

- by means of easy_install
- by downloading it
  - from the source page
  - from CTAN

Through easy_install

With the terminal, do:

```
sudo easy_install dot2tex
```

Installation of dot2tex

There are two ways to install it:

- by means of easy_install
- by downloading it
  - from the source page
  - from CTAN

From source

After having downloaded the .zip file, unpack it, move into the directory and do:

```
sudo python setup.py install
```

Outline

How it works - Outline

The dot code

- Create in your desktop a directory named ex1
- Create with gedit a new empty file called ex1.dot and paste this code:

```
digraph G {
    1->2 [label="1/2"];
    2->3 [label="1/2"];
    3->1 [label="1/2"];
}
```

Obtaining the Ti*k*Z picture

- At this point use the terminal to move into the folder:

```
cd Desktop/ex1/
```

- To create the .tex file use:

```
dot2tex --preproc ex1.dot | dot2tex > ex1.tex
```

- To obtain the image, run:

```
pdflatex ex1.tex
```

The result

- The LaTeX document created by running the dot2tex command is complete
  - it has its own preamble
  - it uses one page just for the picture
  - the result is shown on the right
- Pay attention: the option preproc needs Preview
- You can copy the code of the picture into another document, but be aware of insert also the tikzlibrary reported in the preamble

The result

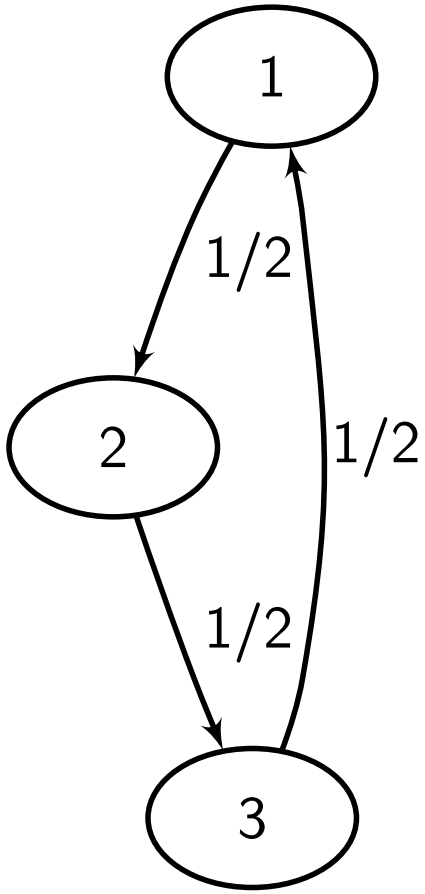
- The LaTeX document created by running the dot2tex command is complete
  - it has its own preamble
  - it uses one page just for the picture
  - the result is shown on the right
- Pay attention: the option preproc needs Preview
- You can copy the code of the picture into another document, but be aware of insert also the tikzlibrary reported in the preamble

The result

- The LaTeX document created by running the dot2tex command is complete
  - it has its own preamble
  - it uses one page just for the picture
  - the result is shown on the right
- Pay attention: the option preproc needs Preview
- You can copy the code of the picture into another document, but be aware of insert also the tikzlibrary reported in the preamble

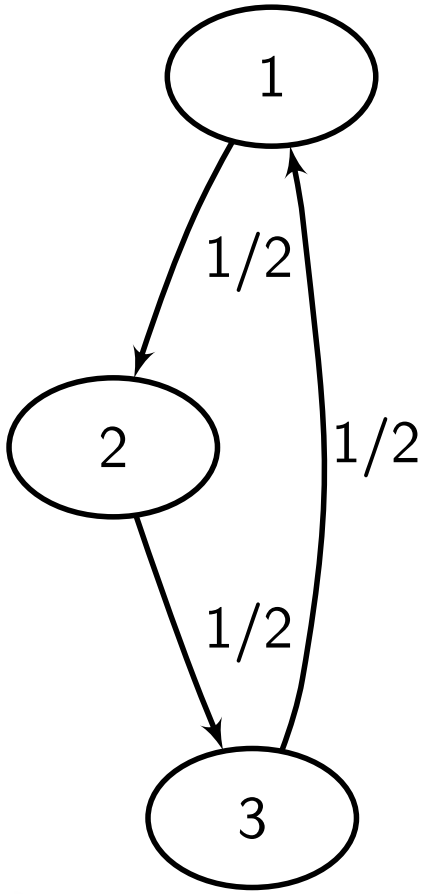
The result

- The LaTeX document created by running the dot2tex command is complete
  - it has its own preamble
  - it uses one page just for the picture
  - the result is shown on the right
- Pay attention: the option preproc needs Preview
- You can copy the code of the picture into another document, but be aware of insert also the tikzlibrary reported in the preamble



The result

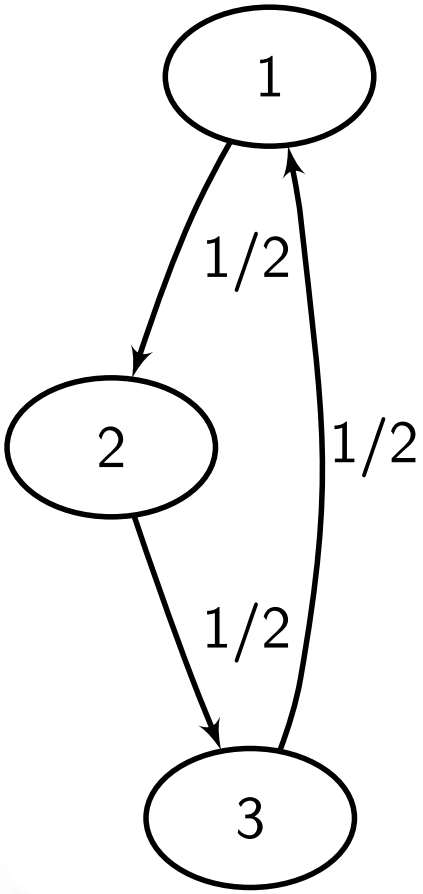
- The LaTeX document created by running the dot2tex command is complete
  - it has its own preamble
  - it uses one page just for the picture
  - the result is shown on the right
- Pay attention: the option preproc needs Preview
- You can copy the code of the picture into another document, but be aware of insert also the tikzlibrary reported in the preamble




```
\usepackage{tikz}
\usetikzlibrary{snakes,arrows,shapes}
```

The result

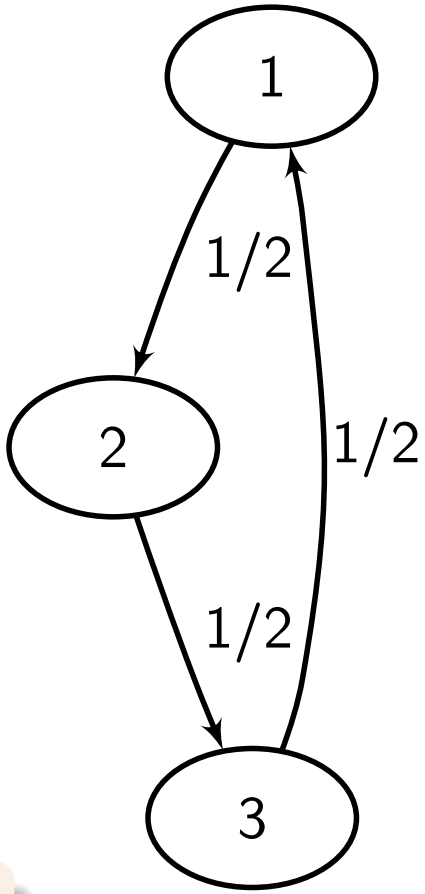
- The LaTeX document created by running the dot2tex command is complete
  - it has its own preamble
  - it uses one page just for the picture
  - the result is shown on the right
- Pay attention: the option preproc needs Preview
- You can copy the code of the picture into another document, but be aware of insert also the tikzlibrary reported in the preamble




```
\usepackage{tikz}
\usetikzlibrary{snakes,arrows,shapes}
```

The result

- The LaTeX document created by running the dot2tex command is complete
  - it has its own preamble
  - it uses one page just for the picture
  - the result is shown on the right
- Pay attention: the option preproc needs Preview
- You can copy the code of the picture into another document, but be aware of insert also the tikzlibrary reported in the preamble




```
\usepackage{tikz}
\usetikzlibrary{snakes,arrows,shapes}
```

How it works - Outline

The output options

- If you do not specify nothing, the picture is written in PGF code; the same result is obtained thanks to the option fpgf
- To have the picture in Ti*k*Z code the option that has to be exploited is ftikz
- The last option is fpst: this force the software to write PSTricks code

Example PGF

```
dot2tex -fpgf ex1.dot > ex1_pgf.tex
```

The output options

- If you do not specify nothing, the picture is written in PGF code; the same result is obtained thanks to the option fpgf
- To have the picture in Ti*k*Z code the option that has to be exploited is ftikz
- The last option is fpst: this force the software to write PSTricks code

Example Ti*k*Z

```
dot2tex -ftikz ex1.dot > ex1_tikz.tex
```

The output options

- If you do not specify nothing, the picture is written in PGF code; the same result is obtained thanks to the option fpgf
- To have the picture in Ti*k*Z code the option that has to be exploited is ftikz
- The last option is fpst: this force the software to write PSTricks code

Example PSTricks

```
dot2tex -fpst ex1.dot > ex1_pst.tex
```

What are the differences?

- PSTricks, to the best knowledge of the author, today is not largely used as Ti*k*Z/PGF
- PGF is much more *machine*-friendly
  - more lines of code are necessary than Ti*k*Z
- Ti*k*Z is *user*-friendly
  - the suggestion is to use it if you want successively modify the picture

How it works - Outline

The dot code

In this second example, we introduce labels also into nodes.

- Create in your desktop a directory named ex2
- Create with gedit a new empty file called ex2.dot and paste this code:

```
digraph G {
    a_1->b_2 [label="1/2"];
    b_2->c_3 [label="1/2"];
    c_3->a_1 [label="1/2"];
}
```

Obtaining the Ti*k*Z picture

In this frame are reported three ways to obtain the picture:

- in *math* mode, the label will be put into $ $: useful to represent the usual math relations of LaTeX

```
dot2tex -tmath ex2.dot > ex2.tex
```

- in *verbatim* mode, the label will not show special LaTeX characters

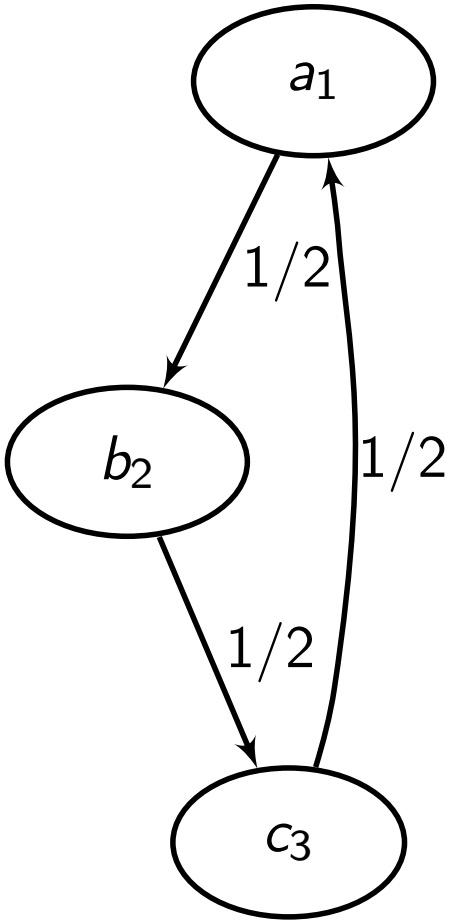
```
dot2tex -tverbatim ex2.dot > ex2.tex
```

- in *raw* mode, the label will not be processed at all

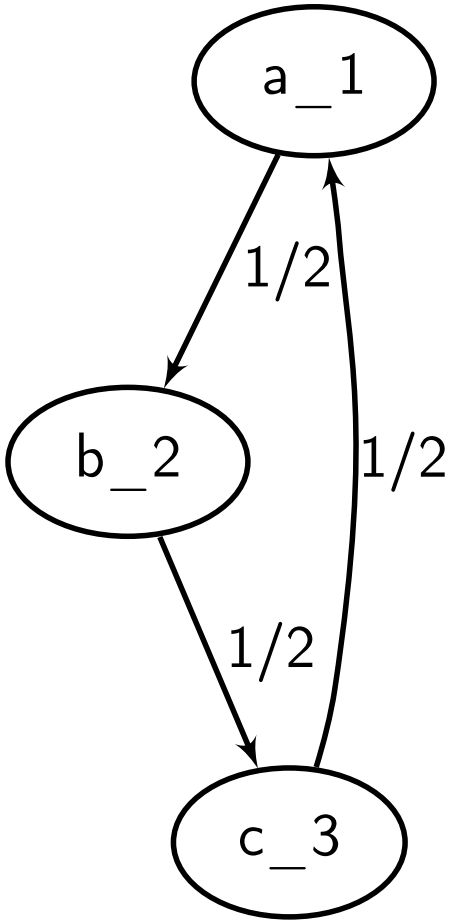
```
dot2tex -traw ex2.dot > ex2.tex
```

The results

Math mode:



Verbatim mode:



Raw mode:

by running **pdflatex** you obtain an error: this is due to the fact that in LaTeX the _ character is a special character and, to be represented needs _

Another method to put labels

It is possible introduce labels for nodes also into the dot code:

```
digraph G {
    1 [texlbl="$a_1$"];
    2 [texlbl="$b_2$"];
    3 [texlbl="$c_3$"];
    1->2 [label="1/2"];
    2->3 [label="1/2"];
    3->1 [label="1/2"];
}
```

In this case, you just need:

```
dot2tex ex2.dot > ex2.tex
```

The mode can be omitted because is assumed to be given inside the texlbl.

How it works - Outline

Label colors

- To insert a colored label, use:

```
digraph G {
  1 [texlbl="$a_1$", lblstyle="red"];
  2 [texlbl="$b_2$"];
  3 [texlbl="$c_3$"];
  1->2 [label="1/2"];
  2->3 [label="1/2"];
  3->1 [label="1/2"];
}
```

The result is:

Label colors

- To insert a colored label, use:

```
digraph G {
  1 [texlbl="$a_1$", lblstyle="red"];
  2 [texlbl="$b_2$"];
  3 [texlbl="$c_3$"];
  1->2 [label="1/2"];
  2->3 [label="1/2"];
  3->1 [label="1/2"];
}
```

The result is:

Label colors

- To insert a colored label, use:

```
digraph G {
  1 [texlbl="$a_1$", lblstyle="red"];
  2 [texlbl="$b_2$"];
  3 [texlbl="$c_3$"];
  1->2 [label="1/2"];
  2->3 [label="1/2"];
  3->1 [label="1/2"];
}
```

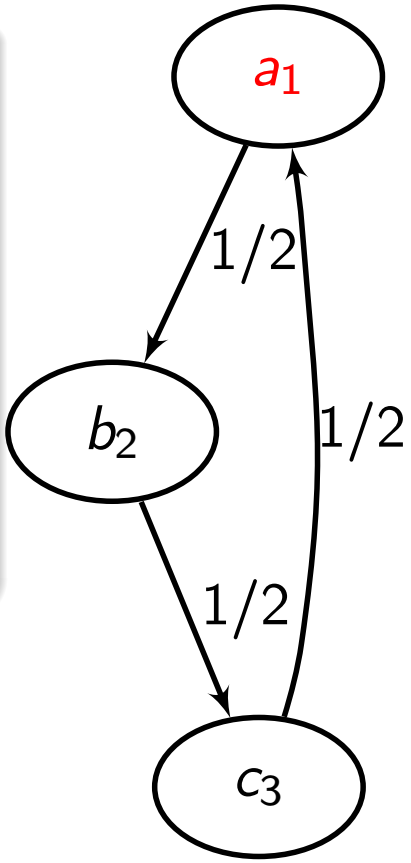
The result is:

Label colors

- To insert a colored label, use:

```
digraph G {
  1 [texlbl="$a_1$", lblstyle="red"];
  2 [texlbl="$b_2$"];
  3 [texlbl="$c_3$"];
  1->2 [label="1/2"];
  2->3 [label="1/2"];
  3->1 [label="1/2"];
}
```

The result is:



Other options

- In general, into the lblstyle it is possible to introduce standard statements of Ti*k*z:

```
digraph G {
1 [texlbl="$a_1$",lblstyle="red"];
2 [texlbl="$b_2$",lblstyle="red"];
3 [texlbl="$c_3$",lblstyle="red"];
1->2 [label="1/2",
      lblstyle="rounded corners,
      fill=blue!20,rotate=30"];
2->3 [label="1/2",
      lblstyle="rounded corners,
      fill=blue!20"];
3->1 [label="1/2",
      lblstyle="rounded corners,
      fill=blue!20, below=0.1cm"];
}
```

The result is:

Other options

- In general, into the lblstyle it is possible to introduce standard statements of Ti*k*z:

```
digraph G {
1 [texlbl="$a_1$",lblstyle="red"];
2 [texlbl="$b_2$",lblstyle="red"];
3 [texlbl="$c_3$",lblstyle="red"];
1->2 [label="1/2",
      lblstyle="rounded corners,
      fill=blue!20,rotate=30"];
2->3 [label="1/2",
      lblstyle="rounded corners,
      fill=blue!20"];
3->1 [label="1/2",
      lblstyle="rounded corners,
      fill=blue!20, below=0.1cm"];
}
```

The result is:

Other options

- In general, into the lblstyle it is possible to introduce standard statements of Ti*k*z:

```
digraph G {
1 [texlbl="$a_1$",lblstyle="red"];
2 [texlbl="$b_2$",lblstyle="red"];
3 [texlbl="$c_3$",lblstyle="red"];
1->2 [label="1/2",
      lblstyle="rounded corners,
      fill=blue!20,rotate=30"];
2->3 [label="1/2",
      lblstyle="rounded corners,
      fill=blue!20"];
3->1 [label="1/2",
      lblstyle="rounded corners,
      fill=blue!20, below=0.1cm"];
}
```

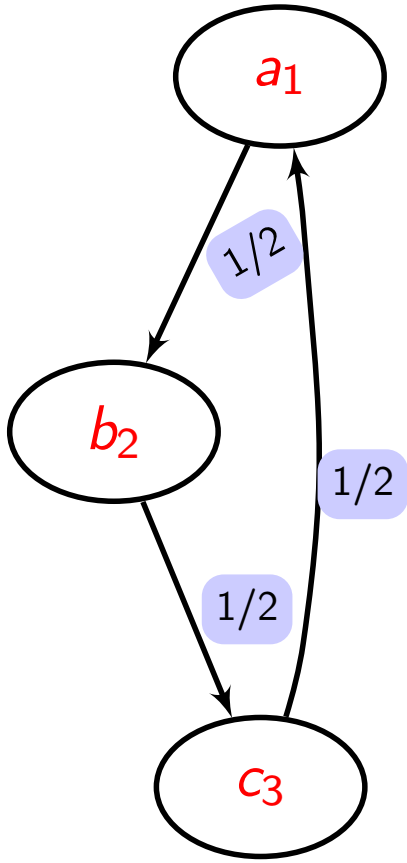
The result is:

Other options

- In general, into the lblstyle it is possible to introduce standard statements of Ti*k*z:

```
digraph G {
1 [texlbl="$a_1$",lblstyle="red"];
2 [texlbl="$b_2$",lblstyle="red"];
3 [texlbl="$c_3$",lblstyle="red"];
1->2 [label="1/2",
      lblstyle="rounded corners,
      fill=blue!20,rotate=30"];
2->3 [label="1/2",
      lblstyle="rounded corners,
      fill=blue!20"];
3->1 [label="1/2",
      lblstyle="rounded corners,
      fill=blue!20, below=0.1cm"];
}
```

The result is:



How it works - Outline

Node characterization

Nodes may have different:

- shapes
  - ellipse
  - circle
  - rectangle
- colours
  - border color
  - fill color
- fonts

There are two ways to characterize nodes:

- global definition (useful to characterize the majority of nodes)
- single definition

An example

- Through the keyword style, nodes are customized with the usual statements of Ti*k*z:

```
digraph G {
    node [style="fill=green!20"];
    1->2 [label="1/2"];
    2->3 [label="1/2"];
    3->1 [label="1/2"];
    3 [shape=rectangle,
      style="fill=cyan!20,draw=blue"]
}
```

the result is:

- By using:

```
dot2tex -ftikz ex3.dot > ex3.tex
```

An example

- Through the keyword style, nodes are customized with the usual statements of Ti*k*z:

```
digraph G {
    node [style="fill=green!20"];
    1->2 [label="1/2"];
    2->3 [label="1/2"];
    3->1 [label="1/2"];
    3 [shape=rectangle,
      style="fill=cyan!20,draw=blue"]
}
```

the result is:

- By using:

```
dot2tex -ftikz ex3.dot > ex3.tex
```

An example

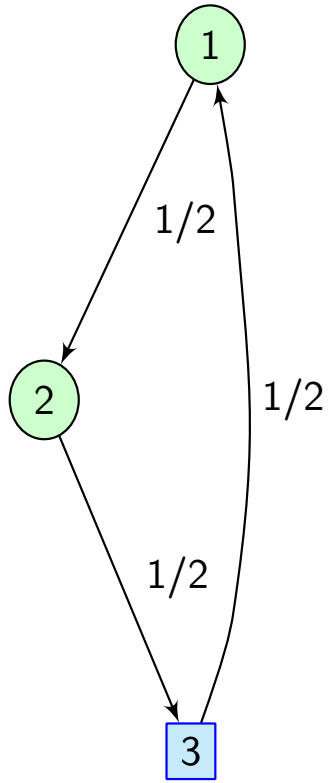
- Through the keyword style, nodes are customized with the usual statements of Ti*k*z:

```
digraph G {
    node [style="fill=green!20"];
    1->2 [label="1/2"];
    2->3 [label="1/2"];
    3->1 [label="1/2"];
    3 [shape=rectangle,
      style="fill=cyan!20,draw=blue"]
}
```

- By using:

```
dot2tex -ftikz ex3.dot > ex3.tex
```

the result is:



An example

- The same code:

```
digraph G {
    node [style="fill=green!20"];
    1->2 [label="1/2"];
    2->3 [label="1/2"];
    3->1 [label="1/2"];
    3 [shape=rectangle,
      style="fill=cyan!20,draw=blue"]
}
```

gives this result:

- By using the default processing (pgf):

```
dot2tex ex3.dot > ex3_pgf.tex
```

An example

- The same code:

```
digraph G {
    node [style="fill=green!20"];
    1->2 [label="1/2"];
    2->3 [label="1/2"];
    3->1 [label="1/2"];
    3 [shape=rectangle,
      style="fill=cyan!20,draw=blue"]
}
```

gives this result:

- By using the default processing (pgf):

```
dot2tex ex3.dot > ex3_pgf.tex
```

An example

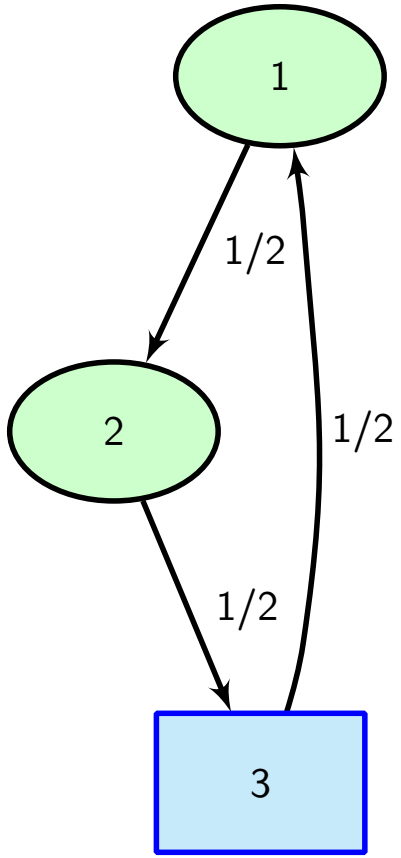
- The same code:

```
digraph G {
    node [style="fill=green!20"];
    1->2 [label="1/2"];
    2->3 [label="1/2"];
    3->1 [label="1/2"];
    3 [shape=rectangle,
      style="fill=cyan!20,draw=blue"]
}
```

- By using the default processing (pgf):

```
dot2tex ex3.dot > ex3_pgf.tex
```

gives this result:



Defining types

- In Ti*k*Z, it is possible to define types for nodes
  - use the **d2tfigpreamble** at the beginning of the dot file
- In the online documentation is not reported the case of plural definition
  - for the best knowledge of the author this feature is fundamental
  - the next example highlight this fact

Example with types

- This graph will have two types of nodes coloured differently

```
digraph G {
    d2tfigpreamble = "\tikzstyle{statecold}= \
    [draw=blue!50,very thick,fill=blue!20],
    \tikzstyle{statehot}= \
    [draw=red!50,very thick,fill=red!20]";
    A [style="statecold"];
    B [style="statecold"];
    C [style="statehot"];
    D [style="statehot"];
    A->B->D;
    A->C->D;
    D->C;
    D->B->A;
    B->B [topath="loop left"];
    C->B;
}
```

Example with types

- Please note the syntax:
  - inside " " the entire definition is reported (both two states in this case)
  - you have to put a \ both in front of **tikzstyle** and before the definition of the type (into [ ]): this is fundamental!
  - you have to separate types with a ,
- The option **topaths** will be subsequently treated

The result (I)

- By using:

```
circo -Txdot ex4.dot | dot2tex -ftikz > ex4.tex
pdflatex ex4.tex
```

- we obtain:

The result (I)

- By using:

```
circo -Txdot ex4.dot | dot2tex -ftikz > ex4.tex
pdflatex ex4.tex
```

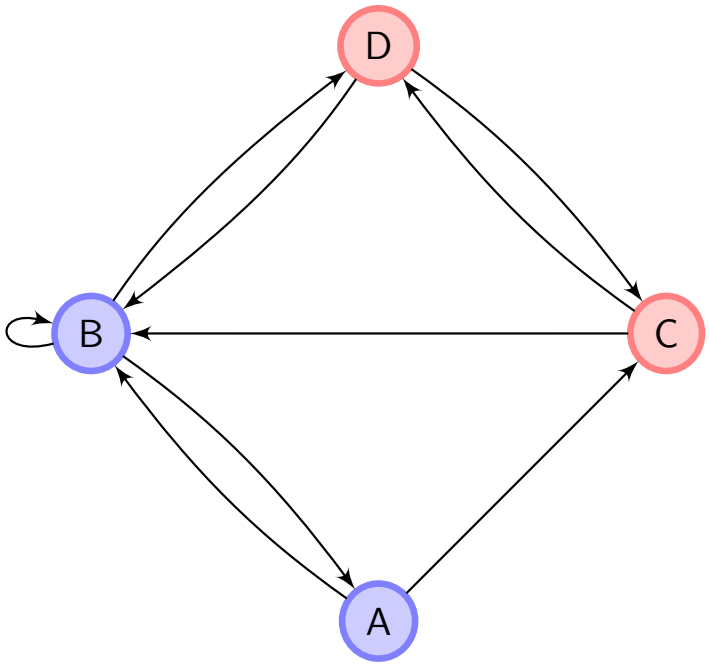
- we obtain:

The result (I)

- By using:

```
circo -Txdot ex4.dot | dot2tex -ftikz > ex4.tex
pdflatex ex4.tex
```

- we obtain:



The result (II)

- By using:

```
circo -Txdot ex4.dot |
   dot2tex -ftikz --styleonly > ex4.tex
pdflatex ex4.tex
```

- we obtain:

The result (II)

- By using:

```
circo -Txdot ex4.dot |
   dot2tex -ftikz --styleonly > ex4.tex
pdflatex ex4.tex
```

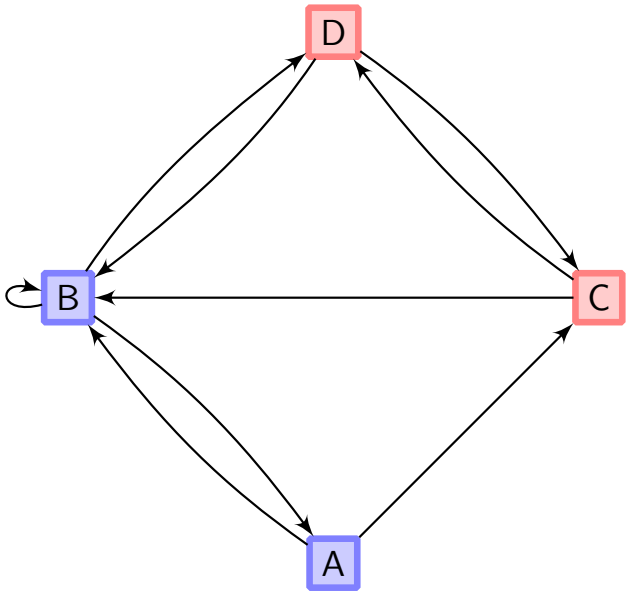
- we obtain:

The result (II)

- By using:

```
circo -Txdot ex4.dot |
   dot2tex -ftikz --styleonly > ex4.tex
pdflatex ex4.tex
```

- we obtain:



The result (III)

- By using:

```
neato -Txdot ex4.dot |
   dot2tex -ftikz --styleonly > ex4.tex
pdflatex ex4.tex
```

- we obtain:

The result (III)

- By using:

```
neato -Txdot ex4.dot |
   dot2tex -ftikz --styleonly > ex4.tex
pdflatex ex4.tex
```

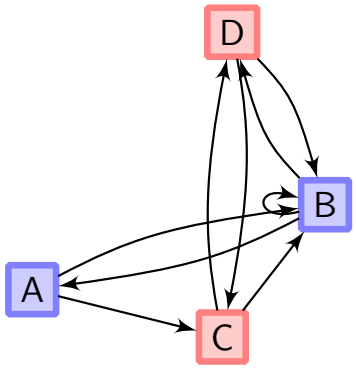
- we obtain:

The result (III)

- By using:

```
neato -Txdot ex4.dot |
    dot2tex -ftikz --styleonly > ex4.tex
pdflatex ex4.tex
```

- we obtain:



Results motivations

- By processing the dot file through circo or neato allows to obtain different kind of results
  - the difference is the layout output
- circo is preferable with regular topologies

How it works - Outline

Edge characterization

- Also edges can be characterized:
  - though the dot syntax:

```
edge [style="..."];
```

  - by means of the option **topath**:

```
a -> b [topath="..."];
```

- with this options, the usual statements of Ti*k*Z can be used

An example: the code

- In this example we exploit the first option:

```
digraph G {
    d2tfigpreamble = "\tikzstyle{statecold}= \
    [draw=blue!50,very thick,fill=blue!20],
    \tikzstyle{statehot}= \
    [draw=red!50,very thick,fill=red!20]";
    A [style="statecold"];
    B [style="statecold"];
    C [style="statehot"];
    D [style="statehot"];
    edge [style="snake=zigzag"];
    A->B->D->C->A;
    edge [style="snake=snake"];
    A->D;
    C->B;
}
```

An example: the result

- By using:

```
circo -Txdot ex5.dot | dot2tex -ftikz -s > ex5.tex
pdflatex ex5.tex
```

- we obtain:

An example: the result

- By using:

```
circo -Txdot ex5.dot | dot2tex -ftikz -s > ex5.tex
pdflatex ex5.tex
```

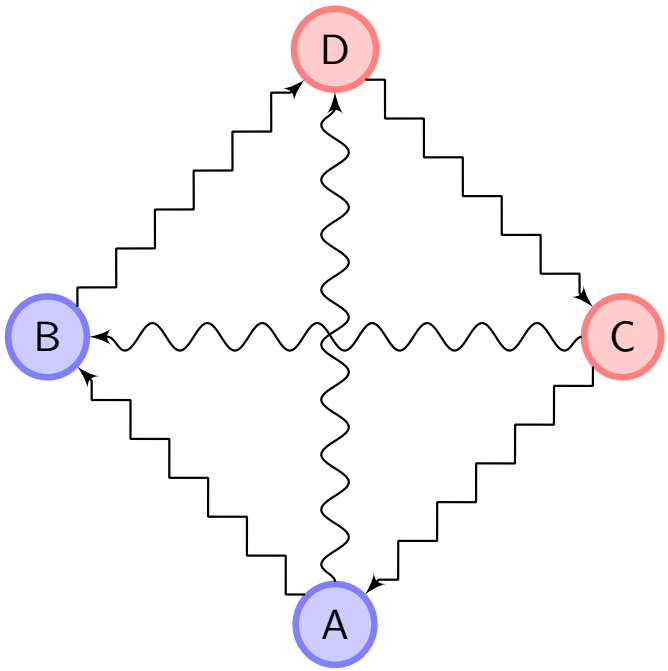
- we obtain:

An example: the result

- By using:

```
circo -Txdot ex5.dot | dot2tex -ftikz -s > ex5.tex
pdflatex ex5.tex
```

- we obtain:



Considerations

- The -s option is needed to manage *snakes*
- The tikzlibrary **snakes** is now deprecated: although the package requires it, should be better exploit the library **decorations**
  - would be interesting because **decorations** allows to exploit *pre/post length* options and other shapes
- It is possible of course change later the library, but you are forced also to insert manually the new options in the Ti*k*Z code

Another example: the code

- In this example we exploit the topath option:

```
digraph G {
    d2tfigpreamble = "\tikzstyle{statecold}= \
    [draw=blue!50,very thick,fill=blue!20],
    \tikzstyle{statehot}= \
    [draw=red!50,very thick,fill=red!20]";
    A [style="statecold"];
    B [style="statecold"];
    C [style="statehot"];
    D [style="statehot"];
    A->B->D->C->A[topath="bend left=20"];
    A->C->D->B->A[topath="bend left=20"];
    B->B [topath="loop left"];
}
```

An example: the result

- By using:

```
circo -Txdot ex6.dot | dot2tex -ftikz > ex6.tex
pdflatex ex6.tex
```

- we obtain:

An example: the result

- By using:

```
circo -Txdot ex6.dot | dot2tex -ftikz > ex6.tex
pdflatex ex6.tex
```

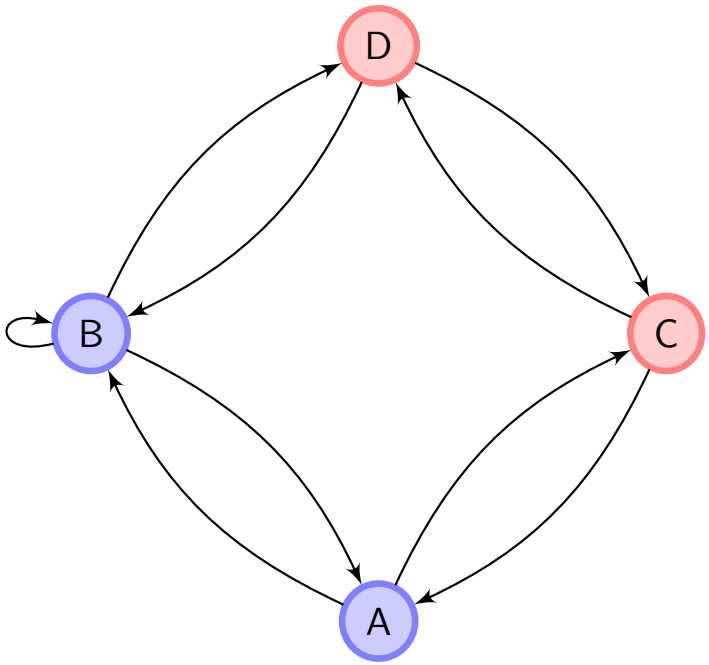
- we obtain:

An example: the result

- By using:

```
circo -Txdot ex6.dot | dot2tex -ftikz > ex6.tex
pdflatex ex6.tex
```

- we obtain:



Outline

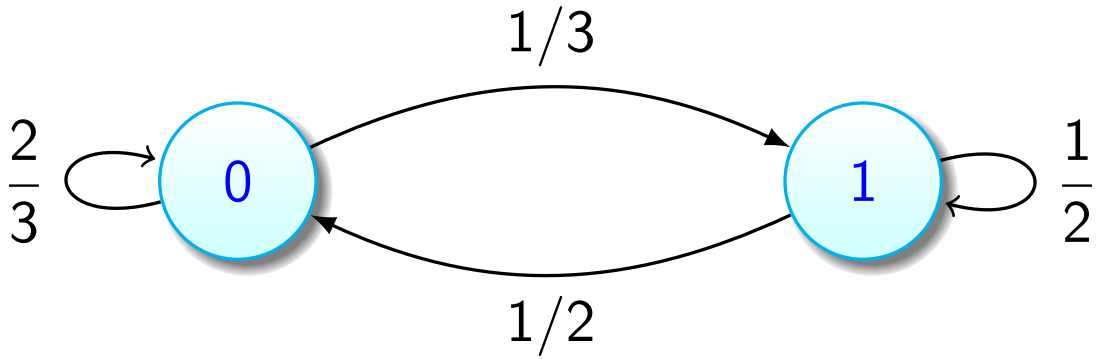
Why use dot?

Someone can claim: why use dot and not directly Ti*k*Z?

- The dot language is much more intuitively
- You can achieve more or less the same quality of Ti*k*Z
- It is more suitable for very large graphs

Where is convenient to use Ti*k*Z?

I think that if you just have to insert a small graph in a document and it has to be created from scratch, the best way to do it is use directly Ti*k*Z. For example:



which has been created thanks to the code reported in the following slide.

Code of previous example

```
\begin{lstlisting}
\begin{tikzpicture}
[-latex,auto,node distance=4cm,on grid,semithick,
state/.style={circle,top color=white,
bottom color=processblue!20,draw, processblue,
circular drop shadow,text=blue,minimum width=1cm}]
\node[state] (A)                {$0$};
\node[state] (B) [ right=of A] {$1$};
\path (A) edge [loop left] node{$\dfrac{2}{3}$} (A);
\path (B) edge [loop right] node{$\dfrac{1}{2}$} (B);
\path (B) edge [bend left=25] node[below]{$1/2$} (A);
\path (A) edge [bend right=-25 ] node[above]{$1/3$} (B);
\end{tikzpicture}
```

Comments

- The main concept is the definition of the *state*: this is similar to the procedure used through the option d2tfigpreamble
- The nodes must be placed in some sense (the position of B is given in terms of A's position): in dot this step is completely avoided
- The syntax to connect nodes is a bit more heavy than dot

Licence and Credit

This presentation:

- is licensed (CC BY-NC-ND 3.0)
- has been realized through Arkyenell Theme licensed (CC BY-NC-SA 3.0)