Petri-nets packages

Franck Pommereau (pommereau@univ-paris12.fr)

Last update: 2002-10-13

Abstract

This paper describes Petri-nets, a set of TEX/LATEX packages about Petri nets and related models. One package allows to draw Petri-nets in PostScript or PDF documents. One other defines macros related to PBC, M-nets and B(PN)^2 models. A last package just gathers together the two previous.

Contents

1 Introduction **2**
- 1.1 Installation 2
- 1.2 Loading the packages 4
- 1.3 Things to do, known bugs and problems 4
- 1.4 Legal stuff 5
- 1.5 Contributors 5

2 Text commands: package `pntext` **5**

3 Drawing nets: package `pndraw` **6**
- 3.1 Some words about PSTricks 6
- 3.2 Beginning and ending nets 9
- 3.3 Drawing nodes 10
  - 3.3.1 Additional labels 11
  - 3.3.2 Free text 12
- 3.4 Linking nodes 12
  - 3.4.1 Additional labels 14
  - 3.4.2 Arcs 14

4 Tips, tricks and troubleshooting **14**

5 Producing PDF documents **16**

A Reference pages **18**
A.1 Text commands: package `pntext` 18
A.2 Drawing command: package `pndraw` 18

1 Introduction

Petri-nets is a set of packages I use to write my papers about Petri nets and related models (essentially: PBC, M-nets and B(PN)^2). It features three packages: one for drawing net, one other which defines additional macros for textual purpose (like B(PN)^2's logo) and a last one for both possibilities. The first package as been designed with a precise goal: saving me a lot of work when I have to draw nets. This goal explains some choices I did: typesetting labels in math mode by default, ending commands with the line, etc. I feel this results in an intuitive and high-level way to define Petri nets, but if you have ideas to improve this, don't hesitate to contact me. The text command package is more classical and if it saves also a lot of typing, its main goal is to ensure uniformity in the notations.

All these packages may evolve and grow up and if you (or I) feel that a feature is missing, you can send me an e-mail explaining your needs (TEX code is welcome too). Notice I don't pretend being a specialist in Petri nets and I'm only used to work on three or four models; so I may ignore particular things in particular models. This may explain some missing features.

If you wish to be notified of the updates of these packages, just send me an email with your request at *pommereau@univ-paris12.fr*.

1.1 Installation

The last version of the package may be downloaded at *http://www.univ-paris12.fr/lacl/pommereau/petrinets.tar.gz*, a copy should be available on *CTAN/macros/generic/petri-nets*; the distribution contains the following files:

- `pnets.tex` is the main TEX package;
- `pnets.sty` is its LATEX counterpart;
- `pndraw.tex` and `pndraw.sty` are the TEX and LATEX sub-packages for drawing nets;
- `pntext.tex` and `pntext.sty` are the TEX and LATEX sub-packages for text commands;

- `pnversion.tex` defines macro `\pnversion`;
- `pndoc.ps` and `pndoc.tex` are the present paper and its source;
- `COPYING` is the text of the GNU GPL;
- `Changelog` collects all changes done to Petri-nets;
- `README` is a short introduction text;
- `pn2pdf` a Perl script used to produce PDF files with pdfLaTeX.

In order to have Petri-nets working, you need to copy files `pnets.*`, `pndraw.*` and `pntext.*` in a place where TeX will be able to find them. This can be somewhere in the default search path (see your local TeX documentation) or in a directory included in your `TEXINPUT` environment variable.

Packages from Petri-nets have been developed and tested with teTeX-0.9 for Linux. TeX packages should work with any TeX version greater than or equal to 3 (my version is 3.14159). LaTeX packages have been designed for LaTeX2$_\varepsilon$ so you may experiment troubles with an older version.

In order to typeset symbols for classical sets of numbers ($\mathbb{N}$, $\mathbb{R}$, etc.) Petri-nets uses fonts from AMS. So you must have package `amsfonts` installed for LaTeX or fonts `bbm` for TeX (I think these conditions are more or less equivalent).

The drawing basis are provided by package PSTricks, my version is PST97 but I used only macros described in the manual of version 0.93a so I hope it should work with it.

Using PSTricks has an important consequence: actual drawings are done in PostScript, so, you may not see them directly from the DVI viewer (at least not correctly); additionally, you must use a PostScript driver in order to produce your final document (for example, dvips works well). Support for producing PDF documents with pdfLaTeX is provided (see section 5).

PSTricks conflicts with packages `graphics` and `graphicx`, Petri-nets thus inherits this conflict. If you wish to use them together, you should first load package `pstcol`, then Petri-nets and at last, `graphics` or `graphicx`. The following is quoted from PSTricks' `README`:

> To use the standard '`color`' package (which is available both for plain TeX and LaTeX) with PSTricks, you must load the '`pstcol`' extra package written by David Carlisle, which interface the two packages, loading them in the right order, and overriding some

small parts of PSTricks to allow it to use the '`color`' package system for specifying color. We *strongly* recommend that you use this way today.

LaTeX users must also take care that the '`pstcol`' package is required in place of the '`pstricks`' one if the '`graphics`' or '`graphicx`' package is also loaded.

If you wish to produce PDF files using pdfLaTeX, you also need to copy the script `pn2pdf` in a directory from which it can be executed. This script requires a working Perl with the package `Digest::MD5` installed. Moreover, the script calls the following programs: `latex`, `dvips` and `epstopdf` which must be installed on your system (there should be not problem for the first two).

You can test your installation by recompiling the present manual which is written in LaTeX. If it compiles successfully, this means Petri-nets and also PSTricks have been found by TeX. If one file is not found, TeX will complain, giving the file name. Then, running dvips will ensure that the PostScript headers of PSTricks can be found.

If you successfully use Petri-nets under another configuration, feel free to send me an e-mail at *pommereau@univ-paris12.fr*. You may also send bug reports or comments, they are welcome. Of course, bug fixes are welcome too.

1.2 Loading the packages

From TeX, you load the package with `\input pnets`, from LaTeX, you should put `\usepackage{pnets}` in your document preamble. Since you may want to use only the drawing macros, you can use `\input pndraw` or from LaTeX: `\usepackage{pndraw}`. Similarly, in order to load only text commands, you can use `\input pntext` or `\usepackage{pntext}`.

Both packages define a macro `\pnversion` which gives the date of the last update of the package. The date is given as a triple of numbers of the form `year-month-day`.

1.3 Things to do, known bugs and problems

In order to draw additional labels at the right position, `pndraw` has to perform some time consuming floating point computing. I plane in the future to perform this directly in PostScript, in order to speed-up TeX compilation stage.

Only transitions are allowed to change their size according to their inner label, it may be nice to have this feature for the other nodes. But it's may be a little bit difficult or quite ugly for some nodes.

I also could add looping arcs. This is useless for Petri nets but it would allow one to use the packages for other purposes, like drawing automata.

The script `pn2pdf` does not handle documents spread over multiple files: if a picture is created in a file which is included, it will be ignored.

In a late future, I'd like to make the package independent of PSTricks, so it would be more portable (but less powerful).

1.4 Legal stuff

1.5 Contributors

I am very grateful to all the persons who contributed to my work, in particular:

Denis Girou, Hanna Klaudel, André Steenveld.

2 Text commands: package `pntext`

This section lists the commands defined in package `pntext`, they are all available from TEX as from LATEX.

Math sets. Command `\mathset{A}` produces symbol $\mathbb{A}$ (and of course it works for any text). Shortcuts are defined: `\setN` for `\mathset{N}` and similarly for `\setZ`, `\setQ`, `\setR` and `\setC`.

Status of places. Commands `\iplace`, `\eplace` and `\xplace` produce respectively characters i, e and x. Command `\placestatus{n}` typesets character n (in the case you would like a new place type).

Operators for communication. Synchronization operator (**sy**) is available through command `\sy`, in math mode, this command is surrounded by additional white space as usual for binary operators. Similar operators are `\rs` for restriction and `\tie` for asynchronous links. Scoping is available with commands `\lscope` and `\rscope` which produce respectively a left and a right double bracket, you may also use `\scope{A}{N}` to produce $[\![A : N]\!]$.

If you use `\lscope` (resp. `\rscope`) without the corresponding `\rscope` (resp. `\lscope`) in the same equation or array cell, TEX will complain about some missing `\right` (resp. `\left`). In order to close (resp. open) a scope without drawing the bracket, you may use macro `\Rscope` (resp. `\Lscope`).

Choice. Choice operator $\square$ is produced with command `\choice`. In math mode, it behaves like any binary operator.

B(PN)^2. The logo of B(PN)^2 is typeset using command `\bpn`. A B(PN)^2 keyword (say **program**) is typeset with `\bpnkw{program}` and for a non-terminal in the syntax (say scope) you may use `\bpnnt{scope}`. Function Mnet is defined as `\mnet`.

Sets of values and variables. *Var* and *Val* are typeset using commands `\Var` and `\Val`. They are better to use than directly the text they typeset because they adjust their spacing in math mode: compare "Var" with "*Var*".

To be continued...

3 Drawing nets: package `pndraw`

3.1 Some words about PSTricks

Reading PSTricks manual could be a good idea, but if you don't want, you should know a few things about it in order to use Petri-nets successfully.

In PSTricks, any point is designated by two coordinates in a grid, centered on a reference point which has coordinates $(0, 0)$. Figure 1 shows an example.

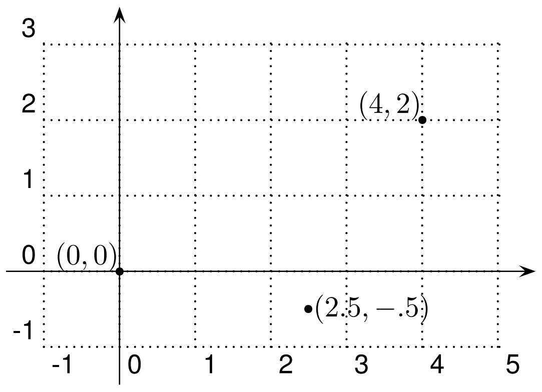


Figure 1: The Cartesian coordinate system.

The distance from the origin $(0, 0)$ to a point (x, y) depends on three parameters. `units` is a global scale, if `unit=1.2cm`, $(4, 2)$ becomes an abbreviation for $(4 \times 1.2 \text{ cm}, 2 \times 1.2 \text{ cm})$. There's also `xunit` and `yunit` which act respectively only on horizontal or vertical scale; for example, if `xunit=20pt` and `yunit=1in`, $(4, 2)$ stands for $(4 \times 20 \text{ pt}, 2 \times 1 \text{ in})$. As you can see on these short examples, `unit`, `xunit` and `yunit` can be affected any value which is a valid TEX `dimen` (or LATEX length).

The default value for all these three parameters is 1 cm; to change it, you can use the `\psset` command as in `\psset{unit=.5cm}` or in a combined way as in `\psset{xunit=1cm,yunit=2cm}`.

Since coordinates are formed as a comma separated couple of numbers, you *must not* use comma as decimal separator here: `(4.5,2.3)` is correct but `(4,5,2,3)` is not.

Sometime you'll be asked to give an angle value as a macro parameter. In Petri-nets, angles are measured in degrees; they can be given as a numerical value or with a one/two letter(s) code as shown on figure 2. Of course, when you specify an angle by its numerical value, you can use any number, even negative, and not only one of the eight values shown on figure 2.

Macro `\psset` is the way to change PSTricks default parameters, it takes one argument which is a comma-separated list of `name=value` pairs. For example, `pndraw.tex` itself uses:

```
\psset{linewidth=.5pt,
       doublesep=.5pt,
       labelsep=2pt}
```

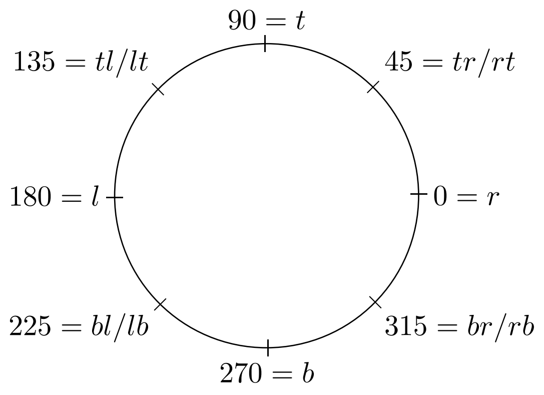


Figure 2: Translation from angle values to letter code. r is for *right*, t for *top*, l for *left* and b for *bottom*.

As you can see, blank spaces before a `name` are ignored. You already know how to change units, some additional `name`s should be useful for Petri-nets:

- `linewidth` is the default width for *any* line drawn by PSTricks;
- `doubleline` can be `true` or `false`, when set to `true`, any line drawn is doubled;
- `doublesep` is the distance between the two lines of a doubled line;
- `labelsep` is the distance between an arc and its label(s);
- `linecolor` is the color used to draw lines, it can be, for example, `black`, `darkgray`, `gray`, `lightgray`, `white`, `red`, `blue`, `green`, `cyan`, `magenta` or `yellow`;
- `fillcolor` is the color used to fill shapes; you can use for it the same colors as for `linecolor`.

PSTricks recognizes many other parameters and you should refer to its manual for all details.

3.2 Beginning and ending nets

To begin a net, just type `\beginnet` for TEX or `\begin{petrinet}` under LATEX; both this commands are expecting two pairs of coordinates to give the bottom-left and upper-right extrema of the net.

For example, `\begin{petrinet}(-1,-2)(5,3)` starts a net which should extend in the rectangular area, which we call the *bounding box*, delimited by $(-1, -2)$ at the lower left corner and $(5, 3)$ at the upper right. These coordinates are sensible to current `unit`, `xunit` and `yunit` values.

The real effect of these two pairs is to fix the size of the bounding box which carry the drawn net (it is actually a `\hbox` in horizontal mode but who cares?). If some stuff in the net extends outside of the declared bounding box, it will be outside and that's all. In other words, the bounding box announced just reserves room for the net and the drawing itself is invisible to TEX. This allows you to lies on the real dimension of your drawings: you can declare a bigger or a smaller bounding box, if you want more or less white space around your nets.

Before to give bounding box coordinates for a net, you can give optional parameters inside square brackets. These parameters are interpreted as PSTricks options which are applied to the following net and stay local to it. For example, if we have `unit=1cm` before starting a net, command `\begin{petrinet}[unit=2cm](0,0)(1,1)` starts a net with `unit=2cm` for its bounding box and any coordinate given inside the net. But as the net ends, the old value `unit=1cm` is restored.

In order to have bounding boxes drawn in your nets: you may use command `\showbb`, optionally followed by bracketed options. For figure 4, I added `\showbb` at the beginning of the net. The starred version of `\showbb` fills its background and so, command `\showbb*[fillcolor=red]` draws a filled red rectangle over the bounding box.

To end the drawing, you should use `\endnet` or `\end{petrinet}` (did you guess?). All macros between these two ones are interpreted as PSTricks or Petri-nets commands. From now, we call a *net* all the stuff inclosed between `\begin{petrinet}` and `\end{petrinet}` (if you use LATEX). One point has to be remembered: inside a net, the end of line has a special meaning because it is used to end most Petri-nets drawing commands. So if you want to use commands on several lines, you have to end every line but the last with a comment (`%`). Also notice that if you call `\psset` command inside a net, its effect will remain local to this particular net.

If you wish something to be done every time a net begins, you may set token list `\everynet` to what you want to be inserted between com-

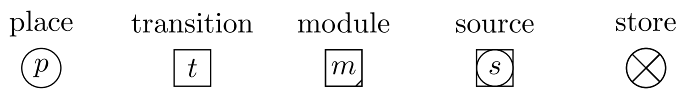


Figure 3: The nodes available in Petri-nets.

mand `\begin{petrinet}` and your drawing commands. For example, with `\everynet={\small}`, all the following nets will be set in small types.

Except for beginning and ending nets, all the macros in the following are the same under TEX or LATEX.

3.3 Drawing nodes

The available node shapes are places, transitions, modules, sources and stores, see figure 3 for their looking. They are produced from macros `\place`, `\trans`, `\module`, `\source` and `\store` respectively. Their size is controlled with dimensions `\placesize`, `\transsize`, `\modulesize`, `\sourcesize` and `\storesize` respectively. For instance, `\transsize=8mm` sets the size of transitions to `8mm`, `\nodessize{5mm}` sets the size of *all* nodes to `5mm`.

Adding a node is made using command "node{name}(x, y)label" where:

- "node" is one of the above node commands;
- "name" is the name of this node which must be unique for a given net (if not, the new node overwrites the old one). This name is case sensitive and should not contain any special characters (as `$`, `\`, etc.);
- "(x, y)" are the coordinates of the node's center;
- "label" is the text to typeset inside the node, it extends until the end of the line. This label is silently discarded for sources.

For instance, the nodes in figure 3 where set with commands:

```
\place{place}(0,0) p
\trans{trans}(1,0) t
\module{module}(2,0) m
\source{source}(3,0) s
\store{store}(4,0) labels are ignored for stores
```

By default, labels are typeset in math mode, if you wish a label typeset in text mode, add option `"` after the node command. Notice that any space after the coordinates of the node is part of the label, it has no importance in math mode but in text mode it leads to a label which starts with a white space. Option `=` sets double-line mode and so the node is drawn with doubled lines. Option `!` sets the line width to 2pt so the node appears thicker than usually. PSTricks options may be given between square brackets, for instance adding `[linecolor=red,linewidth=1pt]` leads to a red node with 1pt lines.

For transitions, an additional option `*` may be used in order to have the boundaries of the transition fitted to the label typeset inside. By default, the size of the nodes is fixed and the label overlaps outside if too long, for transitions, with option `*`, the size is just fine.

3.3.1 Additional labels

You can add additional labels to a node. Each label is typeset using macro `\label`[1] which has two arguments: the first is an angle indication and the second, which extends until the end of the line, is the text to typeset. The angle indication is a numeric value or a code as explained earlier. Option `"` explained above is also available for additional labels (and should be given just after `\label`).

Before I give an example, let me explain a useful parameter for typesetting label: the token list `\everylabel` is expanded before any label is typeset. For example, command `\everylabel={\scriptstyle}` leads to typeset all labels in script style. This works for really any label, not only for nodes labels. If `\everylabel` is set inside a net, its effect will remain local to this net.

The figure 4 has been typeset using the following commands:

```
\begin {petrinet}[xunit=2cm](-.5,-.5)(2.5,.5)
\showbb
\place{p1}(0,0) \bullet
\trans*"{t}(1,0)~transition~
  \label"{70}more text
\place!{p2}(2,0) i
  \label{r} \iplace.\left\{     % I can type multi-line
    {\textstyle 1, 2, 3 \atop   % labels, thanks to comments.
```

[1]For LaTeX, this means that the well known cross-referencing `\label` macro is not available inside a `petrinet` environment. But outside, it works as usual.

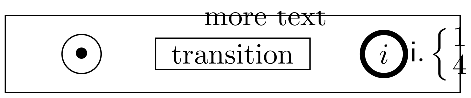


Figure 4: Nodes example. (In the PDF, the outside of the bounding box is cropped, see section 5.)

```
    \textstyle 4, 5, 6} \right\}
\end {petrinet}
```

You may notice that the net is not well centered, it extends more on the right (compare with the caption, which is correctly centered). This is because I declared a bounding box of $(-.5, -.5)$ to $(2.5, .5)$ while in fact, the label on place i extends on the right, outside the bounding box (macro `\showbb` make it visible).

3.3.2 Free text

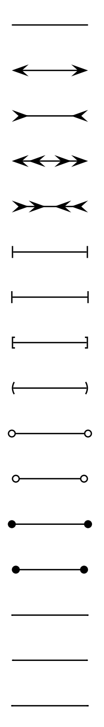
Another kind of node is unboxed text. Macro `\text` is similar to places and transitions macros but it just typesets the given text, centered on the given coordinates. Like for other nodes, the text is typeset in math mode and it can be added labels around with macro `\label` (which takes care of text's bounding box).

This macro knows only option `"` which has the usual meaning.

3.4 Linking nodes

You can draw links between two arbitrary nodes. The package won't check if requested links are valid from Petri nets point of view. Basically, macro `\link` has three parameters: the name of the starting node, that of the ending node and then text to typeset on the link (which extends until the end of the line). You can add optional parameters, before the first node name:

- options inside square brackets are PSTricks options, as usual;
- a real number, between 0 and 1, enclosed in angle brackets (*e.g.*, `<.3>`) can be used to specify the position of the label, between starting and ending nodes. Value 0 stands for "on starting node" while 1 means "on ending node"; the default value is .5 (*i.e.*, at the middle of the link);



`-`	none (default)
`<->`	arrowheads
`>-<`	reverse arrowheads
`<<->>`	double arrowheads
`>>-<<`	reverse double arrowheads
`|-|`	T-bar flush to endpoints
`|*-|*`	T-bar centred on endpoints
`[-]`	square brackets
`(-)`	rounded brackets
`o-o`	circles centred on endpoints
`oo-oo`	circles flush to endpoints
`*-*`	disks centred on endpoints
`**-**`	disks flush to endpoints
`c-c`	extended rounded ends
`cc-cc`	flush rounded ends
`C-C`	extended square ends

Figure 5: The different types of links termination. They can be freely mixed.

- characters `^`, `_` or `*` may be used to specify the side of the link where the label is typeset. For an link which extends from the left to the right, `^` means "above", `_` means "below" and `*` is for "over". If nothing is specified, the label is placed above the link;
- options `"`, `=` and `!` have their usual meanings;
- arrowheads and such can be specified between two `/` signs. The available values are shown in figure 5, default is no arrow.

The order in which you use these options has no importance, provided all options are given before the starting node. Additionally, if you use contradicting options, the last used overrides any previous. For instance, the two following lines are equivalent:

```
\link_[linecolor=gray,linewidth=5pt]*<.2>^[linewidth=1pt]
\link^<.2>[linecolor=gray,linewidth=1pt].
```

To draw curved links, you should use PSTricks' `arcangle` option which is an angle in degrees, measuring the deviation from a straight line between nodes, at the start and at the end of the link. Try, it's easy.

To change the arrowhead appearance, you must change PSTricks' parameters `arrowsize`, `arrowlength` and `arrowinset`. Petri-nets uses the following:

```
\psset{arrowsize=3pt 3,
       arrowlength=1.4,
       arrowinset=.4}
```

I wont explain how it works, just try, it's easy too. (Or read PSTricks manual, all details are given.)

3.4.1 Additional labels

Like for nodes, links can be added more labels. As usual, this is made with macro `\label`, but here, its syntax is like for macro `\link`, except you must not specify starting and ending nodes (but links options are available).

3.4.2 Arcs

Since links are almost always draw with a single arrowhead at the end (and then called *arcs*), a shortcut is provided: macro `\arc` may be used instead of `\link/->/`.

4 Tips, tricks and troubleshooting

Error messages. Sometimes, you'll get an error message such as:

```
! Argument of \net:place:draw has an extra }.
<inserted text>
                \par
l.785 \place{i3}(2,0) i_3

?
```

This usually means that the error was just before the line for which TEX complains (here line 785 which is correct). And usually, this error is an omission of one of the arguments of a drawing macro.

Another kind of message occurs quite often:

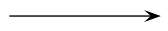


Figure 6: An arc anchored on empty texts

```
PSTricks error.  See User's Guide for further information.
                 Type  H <return>  for immediate help.
! Graphics parameter `rcangle' not defined..
\@pstrickserr ... immediate help.}\errmessage {#1}
                                                  \endgroup
l.794 \arc[rcangle=30]
                      {ab2}{i3} \bullet
?
```

This is because on line 794, you misspelled word "arcangle", giving "rcangle" instead. This error is detected by PSTricks since bracketed arguments are sent verbatim to its macros.

In general, since package `pndraw` uses a lot of tricks in order to read arguments of commands, you should not trust too much error messages: just look at the line indicated in the message and seek for a mistake in the lines before this one. (Actually, this advice can be used for almost any error message issued by LaTeX. . .)

Drawing commands on multiple lines. Inside a net, label arguments are delimited by the end of the line. To type an argument which extends on more than one line, you should end each line but the last with a comment `%`.

Empty node. If you want to draw "floating" arcs, *i.e.*, arcs wich are not attached to a place or a transition, you just have to anchor them on empty texts: the following net is depicted in figure 6.

```
\begin {petrinet}(0,0)(2,0)
\text{from}(0,0)
\text{to}(2,0)
\arc{from}{to}
\end {petrinet}
```

Text mode labels. Take care of the interactions between `\everylabel` and option `"` which asks a label to be typset in text mode. For example, setting `\everylabel{\scriptstyle}` leads to an error every time a text mode label is typeset because `\scriptstyle` is a math command. So you should prefer the long but safe form: `\everylabel{\ifmmode\scriptstyle\fi}` which switches to script-style only if math mode is set.

Labels at the wrong position. If the additional labels you add to a node appear centered on it instead of outside, this may come from the way you visualize your document. DVI viewers usually do not interpret correctly all of the PostScript commands used by PSTricks. Trying a PostScript viewer (GhostView is certainly a good choice) may solve your problem.

To be continued...

5 Producing PDF documents

The LaTeX version of `pndraw` provides support for pdfLaTeX through the Perl script pn2pdf included in the archive. The usage is quite automated, even completely if you run `pdflatex` with the option `--shell`. If you don't want to enable this option, run first "`pn2pdf document`" (assuming your file is called `document.tex`) and then run `pdflatex`. Using option `--shell` simply allows `pdflatex` to run `pn2pdf` for you.

`pn2pdf` produces a file named `document-fig`*i*`.pdf` for each environment `petrinet` found in `document.tex`, where i is 1 for the first found, 2 for the second one, and so on. These files are automatically included during the compilation. A file `document.sum` is also produced in order to remember some information which is used to avoid recreating `-fig`*i*`.pdf` files if not necessary.

There is several important things you should notice:

1. Everything outside of the bounding box of a Petri nets is cropped during the creation of the PDF figure. So, take care to check how your picture is rendered (using `\showbb` for instance).

2. If you want to use PSTricks commands, you should prefix them with `\pst` so they will be properly handled by pdfLaTeX. If you don't use `\pst`, a helpless error message will be issued.

3. Since each picture is rendered separately, all what you define in the document has no effect on the figure (except for the material given

before `\begin{document}` which is always used). You must consider that each picture is a separate document.

4. Only one pass of LaTeX is made on each figure. If you need to several passes, use `pn2pdf` with options `-f` and `-k`.

5. The script `pn2pdf` will generate a picture for each `petrinet` environment found in the document, even if it is included, for instance in a `verbatim` environment. This is why, in this document, examples are typeset `\begin {petrinet}` ··· `\end {petrinet}` in order to avoid the creation of a picture.

6. Notice that LaTeX and pdfLaTeX do not produce identical rendering of the same document. However, Petri-nets pictures should be the same since they are always made by LaTeX.

To conclude, here is a summary of the options accepted by `pn2pdf` if you have to run it yourself:

`-k`, `--keep` don't delete the temporary files.

`-c`, `--clean` delete these files.

`-f`, `--force` recreate pictures even if their source did not change.

`-h`, `--help` print this help.

You may produce pictures for several documents, in this case, each option applies to all the document given subsequently (in this case, `-c` may be useful to override the use of `-k`).

A Reference pages

A.1 Text commands: package `pntext`

`\bpn`	B(PN)^2
`\bpnkw{key-word}`	**key-word**
`\bpnnt{non-terminal}`	non-terminal
`\choice`	$\Box$
`\eplace`	e
`\iplace`	i
`\lscope`	$[\![$
`\Lscope`	invisible version of `\lscope`
`\mathset{A}`	$\mathbb{A}$
`\mnet`	Mnet
`\placestatus{d}`	d
`\pnversion`	2002-10-13 (current version)
`\rs`	**rs**
`\rscope`	$]\!]$
`\Rscope`	invisible version of `\rscope`
`\scope{a}{N}`	$[\![a : N]\!]$
`\setC`	$\mathbb{C}$
`\setN`	$\mathbb{N}$
`\setQ`	$\mathbb{Q}$
`\setR`	$\mathbb{R}$
`\setZ`	$\mathbb{Z}$
`\sy`	**sy**
`\tie`	**tie**
`\Val`	*Val*
`\Var`	*Var*
`\xplace`	x

A.2 Drawing command: package `pndraw`

The parts placed between ⟨angle brackets⟩ are the optional ones, the others are mandatory. Symbol $\hookleftarrow$ denotes the end of the line.

- `\arc`⟨`^_*"=!`[options]<pos>⟩`{`node1`}{`node2`}`⟨label⟩$\hookleftarrow$
  Draws a labelled arc between node1 and node2.

`^`	label above the arc
`_`	label below the arc
`*`	label on the arc
`"`	label in text mode
`=`	double line
`!`	thick line
`[`options`]`	PSTricks options
`<`pos`>`	position of the label ($0 \leq \text{pos} \leq 1$)

- `\beginnet`⟨`[`options`]`⟩$(x_1, y_1)(x_2, y_2)$↩
  `\begin{petrinet}`⟨`[`options`]`⟩$(x_1, y_1)(x_2, y_2)$↩
  Begins a net whose bounding box is defined by (x_1, y_1) as bottom left corner and (x_2, y_2) at top right corner.

`[`options`]`	PSTricks options

- `\endnet`
  `\end{petrinet}`
  Ends a net.

- `\everylabel={`⟨tokens⟩`}`
  Expands tokens each time a label is typeset.

- `\everynet={`⟨tokens⟩`}`
  Expands tokens each time a net is started.

- `\label`⟨`"{`pos`}`label⟩↩
  Draws an additional node label.

`"`	label in text mode
`{`pos`}`	position of the label, an angle (in degrees) or a corner code (t,l,b,r,tl,tr,bl,br)

- `\link`⟨`^_*"=!``[`options`]``<`pos`>``/`arrow`/`⟩`{`node1`}``{`node2`}`⟨label⟩↩
  Draws a labelled link between node1 and node2.

`^`	label above the arc
`_`	label below the arc
`*`	label on the arc
`"`	label in text mode
`=`	double line
`!`	thick line
`[`options`]`	PSTricks options
`<`pos`>`	position of the label ($0 \leq \text{pos} \leq 1$)
`/`arrow`/`	arrows specifications

- `\module`⟨`"=!`[options]⟩`{`name`}`(x, y)⟨label⟩↩
  Draws a labelled module centered on (x, y).

`"`	label in text mode
`=`	double line
`!`	thick line
[options]	PSTricks options

- `\modulesize=`dimen
  Sets the size of the modules.

- `\nodessize{`dimen`}`
  Sets the size of all nodes.

- `\place`⟨`"=!`[options]⟩`{`name`}`(x, y)⟨label⟩↩
  Draws a labelled place centered on (x, y).

`"`	label in text mode
`=`	double line
`!`	thick line
[options]	PSTricks options

- `\placesize=`dimen
  Sets the size of the places.

- `\psset{`name=value⟨`,...`⟩`}`
  Sets PSTricks options.

`arcangle=`angle	angle at the ends of links
`arrowinset=`real	size of arrowheads insets
`arrowlength=`real	length of arrowheads
`arrowsize=`dim integer	size of arrowheads
`doubleline=`boolean	double lines on/off
`doublesep=`dim	distance between double lines
`fillcolor=`color	background color
`labelsep=`dim	distance between labels and nodes
`linecolor=`color	lines color
`linewidth=`dim	lines thickness
`unit=`dim	global scale
`xunit=`dim	horizontal scale
`yunit=`dim	vertical scale

- `\showbb`⟨`*`[options]⟩↩
  Draws the bounding box of a net.

`*`	background filled
[options]	PSTricks options

- `\source`⟨`"=!`[options]⟩`{`name`}`(x,y)⟨label⟩$\hookleftarrow$
  Draws a labelled source centered on (x,y).

`"`	label in text mode
`=`	double line
`!`	thick line
[options]	PSTricks options

- `\sourcesize=`dimen
  Sets the size of sources.

- `\store`⟨`"=!`[options]⟩`{`name`}`(x,y)⟨label⟩$\hookleftarrow$
  Draws a labelled store centered on (x,y).

`"`	label in text mode
`=`	double line
`!`	thick line
[options]	PSTricks options

- `\storesize=`dimen
  Sets the size of stores.

- `\text`⟨`"`⟩`{`name`}`(x,y)⟨label⟩$\hookleftarrow$
  Draws an unboxed labelled node centered on (x,y).

`"`	label in text mode

- `\trans`⟨`*"=!`[options]⟩`{`name`}`(x,y)⟨label⟩$\hookleftarrow$
  Draws a labelled transition centered on (x,y).

`*`	automatic size
`"`	label in text mode
`=`	double line
`!`	thick line
[options]	PSTricks options

- `\transsize=`dimen
  Sets the size of transitions.