



eolang: \LaTeX Package for Formulas and Graphs of EO Programming Language and φ -calculus*

Yegor Bugayenko
yegor256@gmail.com

2025/09/30, 0.19.0

NB! You must run \TeX processor with `-shell-escape` option and you must have [Perl](#) installed. If you omit the `-shell-escape` option, the package will try to use cached files, if they exist. If they don't, compilation will fail. Thus, when you must prepare your document for a compilation without the `-shell-escape` option, run it locally with the option provided and then package all files (including the files in the `_eolang-*` directories) into a single ZIP archive. It is advised to use `tmpdir` package option in this case, in order to make the directory name not depend on the \TeX engine.

If `-shell-escape` is set, this package won't work on Windows, because it uses POSIX command line interface.

1 Introduction

This package helps you print formulas of φ -calculus, which is a formal foundation of [EO](#) programming language. The calculus was introduced by **bugayenko2021eolang** and later formalized by **kudasov2021**. Here is how you render a simple expression:

*The sources are in GitHub at [objectionary/eolang.sty](#)

$ \begin{aligned} \text{app} &\mapsto \llbracket \\ &\quad \rho \mapsto \xi.b.^2, \alpha_0 t \rightsquigarrow \perp RUE, \\ &\quad b \mapsto \llbracket \alpha_* \mapsto \Phi.\text{fn}(56), \\ &\quad \quad \varphi \mapsto \dot{\Phi}.\text{string.trim}(\xi), \\ &\quad \quad \Delta.. > 01\text{-FE-C3} \rrbracket, \\ &\quad x \mapsto \llbracket \lambda.. > \emptyset \rrbracket. \end{aligned} $	<pre> 1 \documentclass{minimal} 2 \usepackage{eolang} 3 \begin{document} 4 \begin{phiquation*} 5 app -> [[% it's abstract! 6 ^ -> \$.b.^{~2}, 0/t~> TRUE, 7 b -> [[*-> Q.fn(56), 8 @ -> QQ.string.trim(\$), 9 D> 01-FE-C3]]],\ 10 x -> [[\lambda ..> ?]]. 11 \end{phiquation*} 12 \end{document} </pre>
--	---

`phiquation (env.)` The environment `phiquation` lets you write a φ -calculus expressions using simple plain-text notation, where:

- “@” maps to “ φ ” (`\varphi`),
- “^” maps to “ ρ ” (`\rho`),
- “\$” maps to “ ξ ” (`\xi`),
- “?” maps to “ \emptyset ” (`\varnothing`),
- “T” maps to “ \perp ” (`\bot`),
- “Q” maps to “ Φ ” (`\Phi`),
- “QQ” maps to “ $\dot{\Phi}$ ” (`\dot{\Phi}`),
- “->” maps to “ \mapsto ” (`\mapsto`),
- “~>” maps to “ \rightsquigarrow ” (`\rightsquigarrow`),
- “D>” maps to “ $\Delta \mapsto$ ” (`\Delta \mapsto`),
- “L>” maps to “ $\lambda \mapsto$ ” (`\lambda \mapsto`),
- “[[” maps to “ \llbracket ” (`\llbracket`),
- “]]” maps to “ \rrbracket ” (`\rrbracket`),
- “==” maps to “ \equiv ” (`\equiv`),
- “|abc|” maps to “ abc ” (`\text{abc}`).

Also, a few symbols are supported for φ PU architecture:

- “<<” maps to “ \langle ” (`\langle`),
- “>>” maps to “ \rangle ” (`\rangle`),
- “-abc>” maps to “ $\xrightarrow{\text{ABC}}$ ” (`\xrightarrow{\text{ABC}}`),
- “:=” maps to “ \vDash ” (`\vDash`).

Before any arrow you can put a number, which will be rendered as `\alpha` with an index, for example `\phiq{0->x}` will render “ $\alpha_0 \mapsto x$ ”. Instead of a number you can use asterisk too.

You can append a slash and a title to the number of an attribute, such as `0/g->x`. This will render as $\alpha_0|g \mapsto x$. You can use fixed-width words too, for example

`\phiq{0|f|->x}` will render as “ $\alpha_0|f \mapsto x$ ”. It’s also possible to use an asterisk instead of a number, such that `\phiq{*|g->x}` renders as “ $\alpha_*|g \mapsto x$ ”

Numbers are automatically converted to fixed-width font, no need to always decorate them with vertical bars.

TRUE and FALSE are automatically converted to fixed-width font too.

Object names are automatically converted to fixed-width font too, if they have more than one letter.

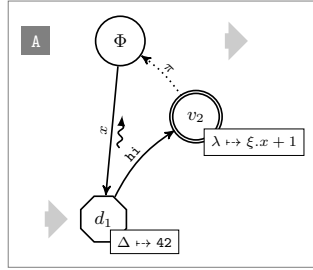
Texts in double quotes are automatically converted to fixed-width font too.

`\phiq` The command `\phiq` lets you inline a φ -calculus expressions using the same simple plain-text notation. You can use dollar sign directly too:

A simple object $x \mapsto \llbracket \varphi \mapsto y \rrbracket$
is a decorator of the data object
 $y \mapsto \llbracket \Delta.. > 42 \rrbracket$.

```
4 \begin{document}
5 A simple object
6 \phiq{x -> [[@ -> y]]} \\
7 is a decorator of
8 the data object \\
9 $y -> [[\Delta ..> 42]]$.
10 \end{document}
```

`sodg (env.)` The environment `sodg` allows you to draw a [SODG](#) graph:



```
1 \documentclass{standalone}
2 \usepackage{eolang}
3 \begin{document}
4 \begin{sodg}
5 v0 \\ v0==> \\ v0!!A
6 v1 xy:v0,-.8,2.8 data:42 tag:d_1
7 v0->v1 a:x rho \\ =>v1
8 v2 xy:v0,+1,+1 atom:\xi.x+1
9 v1->v2 a:|hi| bend:-15
10 v2->v0 pi bend:10 % a comment
11 \end{sodg}
12 \end{document}
```

The content of the environment is parsed line by line. Markers in each line are separated by a single space. The first marker is either a unique name of a vertex, like “v1” in the example above, or an edge, like “v0->v1.” All other markers are either unary like “rho” or binary like “atom:\$\xi.x+1\$.” Binary markers have two parts, separated by colon.

The following markers are supported for a vertex:

- “tag: <math>” puts a custom label <math> into the circle;
- “data: [<box>]” makes it a data vertex with an optional attached “<box>” (the content of the box may only be numeric data);
- “atom: [<box>]” makes it an atom with an optional attached “<box>” (the content of the box is a math formula);
- “box: <txt>” attaches a “<box>” to it;
- “xy: <v>, <r>, <d>” places this vertex in a position relative to the vertex “<v>,” shifting it right by “<r>” and down by “<d>” centimetres;
- “+ : <v>” makes a copy of an existing vertex and all its kids;

- “`edgeless`” removes the border from the vertex;
- “`style:{...}`” adds this TikZ style to the vertex `\node`.

The following markers are supported for an edge:

- “`rho`” places a backward snake arrow to the edge,
- “`bend:<angle>`” bend it right by the amount of “`<angle>`,”
- “`a:<txt>`” attaches label “`<txt>`” to it,
- “`pi`” makes it dotted, with π label;
- “`style:{...}`” adds this TikZ style to the edge `\path`.

It is also possible to put transformation arrows to the graph, with the help of “`v0=>v1`” syntax. The arrow will be placed exactly between two vertices. You can also put an arrow from a vertex to the right, saying for example “`v3=>`”, or from the left to the vertex, by saying for example “`=>v5`.” If you want the arrow to stay further away from the vertex than usual, use a few “`=`” symbols, for example “`==>v0`.”

You can also put a marker at the left side of a vertex, using “`v5!A`” syntax, where “`v5`” is the vertex and “`A`” is the text in the marker. They are useful when you put a few graphs on a picture explaining how one graph is transformed to another one and so forth. You can make the distance between the vertex and the marker a bit larger by using a few exclamation marks, for example “`v5!!!A`” will make a distance three times bigger.

You can make a clone of an existing vertex together with all its dependants, by using this syntax: “`v0+a`.” Here, we make a copy of “`v0`” and call it “`v0a`.” See the example below.

Be aware, unrecognized markers are simply ignored, without any error reporting.

`\eolang` There is also a no-argument command `\eolang` to help you print the name of EO language. It understands the anonymous package option and prints itself differently, to `\phic` double-blind your paper. There is also `\phic` command to print the name of φ -calculus, also sensitive to anonymous mode. The macro `\xmirl` prints “XMIR”.

In our research we use XYZ,
an experimental object-oriented
dataflow language, α -calculus, as its
formal foundation, and XML⁺ —
its XML-based representation.

```

3 \usepackage[anonymous]{eolang}
4 \begin{document}
5 In our research we use \eolang{,}, \
6 an experimental object-oriented \
7 dataflow language, \phic{,}, as its \
8 formal foundation, and \xmirl{ --- \
9 its XML-based representation.
10 \end{document}

```

Without the anonymous option there will be no orange color:

In our research we use EO,
an experimental object-oriented
dataflow language, φ -calculus, as its
formal foundation, and XMIR —
its XML-based representation.

```

3 \usepackage{eolang}
4 \begin{document}
5 In our research we use \eolang{,}, \
6 an experimental object-oriented \
7 dataflow language, \phic{,}, as its \
8 formal foundation, and \xmirl{ --- \
9 its XML-based representation.
10 \end{document}

```

`\phiWave` A few simple commands are defined to help you render arrows.

If x is an identifier and y is an object, then $x \rightsquigarrow y$ makes it a decoratee of an arbitrary number of objects.

```
6 If  $x$  is an identifier and  $y$  is
7 an object, then  $x \phiWave y$ 
8 makes it a decoratee
9 of an arbitrary number of objects.
10 \end{document}
```

`\phi0set` If you want to put a text over an arrow or under it, use `\phi0set` and `\phiUset`
`\phiUset` respectively:

When the names of attributes and their values don't matter, we use an arrow with a star, for example:

$$[\mapsto^*]$$

```
6 When the names of attributes and their
7 values don't matter, we use an arrow
8 with a star, for example:
9 \begin{phiqutation*}
10 [[ \phi0set{*}{->} ]]
11 \end{phiqutation*}
```

`\phiMany` Sometimes you may need to simplify the way you describe an object (the typesetting is a bit off, but this is not because of us, but rather because of [this](#)):

The expression $[\alpha_1 \mapsto x_1, \alpha_2 \mapsto x_2, \dots, \alpha_n \mapsto x_n]$ and expression $[\alpha_i \mapsto_{i=1}^n x_i]$ are syntactically different but semantically equivalent.

```
6 The expression
7 \phiq{[[ 1-> x_1,
8 2-> x_2, \dots,
9 \alpha_n -> x_n ]]}
10 and expression
11 \phiq{[[ \alpha_i
12 \phiMany{->}{i=1}{n} x_i ]]}
13 are syntactically different but
14 semantically equivalent.
```

`\phiSaveTo` If you want to use `phiqutation` or `sodg` environments inside `tabular` or any other
`\sodgSaveTo` environment or command, you won't be able to do this, because `phiqutation` and `sodg` are "verbatim" environments. `\phiSaveTo` and `\sodgSaveTo` commands will help you in this situation. You use them right before `\begin{phiqutation}` or `\begin{sodg}` respectively — the content of the equation or the graph won't be rendered, but instead saved to the file. Later, inside `tabular`, you can use it through the `\input` macro (don't forget the `\parbox`):

Free: $[x \mapsto \emptyset]$

Bound: $[x \mapsto [\Delta.. > 42]]$

```
5 \phiSaveTo{a}
6 \begin{phiqutation*}
7 [[ x -> [[D>42]] ]]
8 \end{phiqutation*}
9 \begin{tabular}{p{.5in}l}
10 Free: &  $[[x \rightarrow ?]]$  \\
11 Bound: &  $\parbox{1in}{\input{a}}$  \\
12 \end{tabular}
```

`\eoAnon` You may want to hide some of the content with the help of the anonymous package option. The command `\eoAnon` may help you with this. It has two parameters: one mandatory and one optional. The mandatory one is the content you want to show and the optional one is the substitution we will render if the anonymous package option is

set.

2 Package Options

`tmpdir` The default location of temp files is `_eolang`. You can change this with the help of the `tmpdir` package option:

```
\usepackage[tmpdir=/tmp/foo]{eolang}
```

`nodollar` You may disable the special treatment of the dollar sign by using the `nodollar` package option:

```
\usepackage[nodollar]{eolang}
```

`anonymous` You may anonymize `\eolang`, `\xmir`, and `\phic` commands by using `anonymous` package option (they all use the `\eoAnon` command mentioned earlier):

```
\usepackage[anonymous]{eolang}
```

`noshell` You may prohibit any interactions with the shell by using the `noshell` option. This may be helpful when you send your document for outside processing and want to make sure the compilation won't break due to shell errors:

```
\usepackage[noshell]{eolang}
```

3 More Examples

The `phiquation` environment treats ends of line as signals to start new lines in the formula. If you don't want this to happen and want to parse the next line as a continuation of the current line, you can use a single backslash as it's done here:

$\frac{x \mapsto [\varphi \mapsto y] \quad y \mapsto [z \mapsto 42]}{x.z \mapsto 42} R1$	<pre> 6 \begin{phiquation*} 7 \dfrac \ 8 {x->[[@->y]] \quad y->[[z->42]]} \ 9 {x.z -> 42} \ 10 \text{\sffamily R1} 11 \end{phiquation*} </pre>
--	---

This is how you can use `\dfrac` from [amsmath](#) for large inference rules, with the help of `\begin{split}` and `\end{split}`:

$\frac{x \mapsto [\varphi \mapsto y, z \mapsto 42, \alpha_0 g \mapsto \emptyset, \alpha_1 \text{foo} \mapsto 42]}{x \mapsto [\varphi \mapsto y, z \mapsto \emptyset, f \rightsquigarrow \text{pi}(\alpha_0 \mapsto [\psi \mapsto \text{hello}(12)], \alpha_1 \mapsto 42)]} R2.$	<pre> 6 \begin{phiquation*} 7 \dfrac{\begin{split} 8 x->[[@->y, z->42, 9 0/g->?, 1/foo->42]] \end{split}}{\begin{split} 10 \end{split}}{\begin{split} 11 x->[[@->y, z->?, f \rightsquigarrow pi (12 0->[[\psi -> hello (12)]], 13 1->42)]] \end{split}}\text{R2}. 14 \end{split}}{\begin{split} 15 \end{split}}\end{phiquation*} </pre>
---	--

You can use the `matrix` environment too, in order to group a few lines:

$\text{foo} \mapsto \left\{ \begin{array}{c} \emptyset \\ \llbracket \lambda.. > \rho \times \xi.\alpha_0 \rrbracket \\ \llbracket \Delta.. > 42 \rrbracket \end{array} \right\}$	<pre> 5 \begin{phiuation*} 6 foo -> \left\{\begin{matrix} \backslash 7 ? \backslash 8 [[L> ~ \times \$. \alpha_0]] \backslash 9 [[D> 42]] \backslash 10 \end{matrix}\right\} 11 \end{phiuation*} </pre>
---	--

The cases environment works too:

$\beta \models \left\{ \begin{array}{l} [v_2, \varphi \xrightarrow{\text{DTZD}} 42] \\ [v_{33}] \end{array} \right.$	<pre> 5 \begin{phiuation*} 6 \beta := \begin{cases} \backslash 7 [v_2, @ -dtzd> 42] \backslash 8 [v_{33}] \backslash 9 \end{cases} 10 \end{phiuation*} 11 \end{document} </pre>
--	--

The phiuation environment may be used together with the [acmart](#) package:

$\begin{array}{l} x \mapsto \llbracket \\ \quad y \mapsto \llbracket \\ \quad \quad z \mapsto \xi, f.. > \emptyset \rrbracket \rrbracket, \\ \beta_1 \models [\psi \xrightarrow{\text{WAIT}} \emptyset]. \end{array}$	<pre> 1 \documentclass{acmart} 2 \usepackage{eolang} 3 \thispagestyle{empty} 4 \begin{document} 5 \begin{phiuation*} 6 x -> [[7 y -> [[8 z -> \$, f ..> ?]]]],\backslash 9 \beta_1 := [\psi -wait> ?]. 10 \end{phiuation*} 11 \end{document} </pre>
---	---

It's possible to use `\label` inside the phiuation environment (pay attention to how you can disable our custom parsing of math formulas by means of curled brackets around the “4” number):

<div style="border: 1px solid black; padding: 10px;"> <p>Discriminant can be calculated using the following simple formula:</p> $D = b^2 - 4ac. \quad (1)$ <p>Eq. 1 is also widely used in number theory and polynomial factoring.</p> </div>	<pre> 6 Discriminant can be calculated using 7 the following simple formula: 8 \begin{phiuation} 9 D = b^{^2} - {4}ac. 10 \label{d} 11 \end{phiuation} 12 Eq.\ref{d} is also widely used in 13 number theory and polynomial factoring. </pre>
--	---

You can add comments to your equations, using the `&\&` command (pay attention, the text inside `\text{}` is not processed and treated like a plain text):

$\llbracket \alpha_0 \mapsto x \rrbracket$	This is formation	6	<code>\begin{phiquation*}</code>
$\llbracket \alpha_0 \mapsto \emptyset \rrbracket$	Abstraction	7	<code>[[0->x]] && \text{This is formation}</code>
$x(\Delta.. > 42)$	Application	8	<code>[[0->?]] && \text{Abstraction}</code>
		9	<code>x(D>42) && \text{Application}</code>
		10	<code>\end{phiquation*}</code>

If you don't use `nodollar` package option, you can still use normal parsing of the dollar sign, by means of `\(...\)` syntax:

The object formation $\llbracket \alpha_0 \mapsto x \rrbracket$ may be replaced with a formula $Q \times a^2$.	6	<code>The object formation $\\$[[0->x]]\\$</code>
	7	<code>may be replaced with a formula</code>
	8	<code>\(Q \times a^2 \).</code>

The `phiquation` environment will automatically align formulas by the first arrow, if there are only left-aligned formulas:

$x(\pi) \mapsto [\lambda.. > f_1],$	5	<code>\begin{phiquation*}</code>
$x(a,b,c) \mapsto [\alpha_0 \mapsto \emptyset, \varphi \mapsto \text{hello}(\xi), x \mapsto \text{FALSE}],$	6	<code>x(\pi) -> [[\lambda.. > f_1]], \\\</code>
$\Delta = 43-09,$	7	<code>x(a,b,c) -> [[\alpha_0 -> ?, \</code>
$x(y) \equiv x(\alpha_0 \mapsto y).$	8	<code>@ -> hello (\$), x -> FALSE]], \\\</code>
	9	<code>\Delta = 43-09 ,</code>
	10	<code>x(y) == x(0-> y).</code>
	11	<code>\end{phiquation*}</code>

If not a single line is indented in `phiquation`, all formulas will be centered:

$\llbracket b \mapsto \emptyset \rrbracket,$	5	<code>\begin{phiquation*}</code>
$\llbracket \varphi \mapsto \perp RUE, \Delta.. > 42 \rrbracket,$	6	<code>[[b -> ?]],</code>
$\psi = \langle \pi, 42 \rangle.$	7	<code>[[@ -> TRUE, \Delta ..> 42]], \\\</code>
	8	<code>\psi = << \pi, 42 >>.</code>
	9	<code>\end{phiquation*}</code>

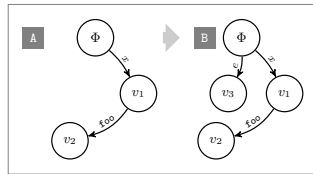
It is possible to use “manual splitting” mode in the `phiquation` environment by starting the body with `\begin{split}`:

$x(\pi) \mapsto 4$	5	<code>\begin{phiquation*}</code>
$x(a,b,c) \mapsto \llbracket \alpha_0 \mapsto \emptyset \rrbracket$	6	<code>\begin{split}</code>
	7	<code>x(\pi) &-> 4 \\\</code>
	8	<code>x(a,b,c) &-> [[\alpha_0 -> ?]]</code>
	9	<code>\end{split}</code>
	10	<code>\end{phiquation*}</code>

When necessary to use a percentage sign, prepend it with a backward slash:

$x \mapsto \text{sprintf}(\text{"Hello, \%s!"}, \text{name})$	5	<code>\begin{phiquation*}</code>
	6	<code>x -> \text{sprintf}(\text{"Hello, \%s!"}, \text{name})</code>
	7	<code>\end{phiquation*}</code>
	8	<code>\end{document}</code>

You can make a copy of a vertex together with its kids:

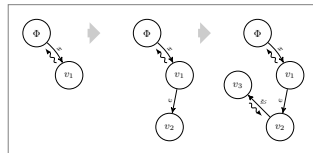


```

5 \begin{sodg}
6 v0 \\\ v0!!A
7 v1 xy:v0,.7,1
8 v0->v1 a:x bend:-10
9 v2 xy:v1,-1.3,.8
10 v1->v2 a:|foo| bend:-20
11 v0+a xy:v0,3,0
12 v3a xy:v0a,-.7,1
13 v0a->v3a a:e bend:-15
14 v0=>v0a \\\ v0a!B
15 \end{sodg}

```

You can make a copy from a copy:

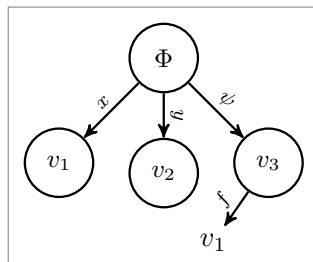


```

5 \begin{sodg}
6 v0
7 v1 xy:v0,.7,1
8 v0->v1 a:x bend:-10 rho
9 v0+a xy:v0,3,0 \\\ v0=>v0a
10 v2a xy:v1a,-.8,1.3
11 v1a->v2a a:e
12 v0a+b xy:v0a,3,0 \\\ v0a=>v0b
13 v3b xy:v2b,-1,-1
14 v2b->v3b a:\psi{} rho
15 \end{sodg}

```

You can have “broken” edges, using “break” attribute of an edge. The attribute must have a value, which is the percentage of the path between vertices that the arrow should take (can’t be more than 80 and less than 20). This may be convenient when you can’t fit all edges into the graph, for example:

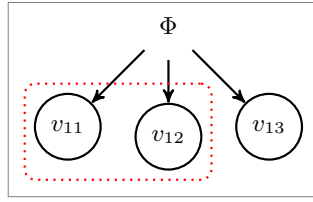


```

5 \begin{sodg}
6 v0
7 v1 xy:v0,-1,1
8 v0->v1 a:x
9 v2 xy:v0,0,1
10 v0->v2 a:y
11 v3 xy:v0,1,1
12 v0->v3 a:\psi{}
13 v3->v1 a:f bend:-75 break:30
14 \end{sodg}

```

You can add [TikZ](#) commands to sodg graph, for example:

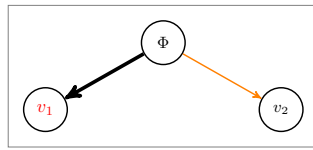


```

6 \begin{sodg}
7 v0 edgeless
8 v11 xy:v0,-1,1 \\\ v0->v11
9 v12 xy:v0,0,1 \\\ v0->v12
10 v13 xy:v0,1,1 \\\ v0->v13
11 \node[draw=red,rounded corners,\
12   dotted,fit=(v11) (v12)] {};
13 \end{sodg}

```

You can modify TikZ style yourself (make sure `style:` stays at the end of the line!), for example:



```

6 \begin{sodg}
7 v0
8 v1 xy:v0,-2,1 style:font=\color{red}
9 v2 xy:v0,2,1
10 v0->v1 style:line width=2pt
11 v0->v2 style:draw=orange
12 \end{sodg}

```

4 Implementation

First, we include a few packages. We need [stmaryrd](#) for `\llbracket` and `\rrbracket` commands:

```
1 \RequirePackage{stmaryrd}
```

We need [amsmath](#) for `equation*` environment:

```
2 \RequirePackage{amsmath}
```

We need [amssymb](#) for `\varnothing` command. We disable `\Bbbk` because it may conflict with some packages from [acmart](#):

```
3 \let\Bbbk\relax\RequirePackage{amssymb}
```

We need [fancyvrb](#) for `\VerbatimEnvironment` command:

```
4 \RequirePackage{fancyvrb}
```

We need [iexec](#) for executing Perl scripts:

```
5 \ifdefined\eolang@noshell\else\RequirePackage{iexec}\fi
```

Then, we process package options:

```

6 \RequirePackage{pgfopts}
7 \RequirePackage{ifluatex}
8 \RequirePackage{ifxetex}
9 \pgfkeys{
10   /eolang/.cd,
11   tmpdir/.store in=\eolang@tmpdir,
12   tmpdir/.default=_eolang\ifxetex-xe\else\ifluatex-lua\fi,
13   nocomments/.store in=\eolang@nocomments,
14   anonymous/.store in=\eolang@anonymous,
15   noshell/.store in=\eolang@noshell,
16   tmpdir
17 }
18 \ProcessPgfPackageOptions{/eolang}

```

Then, we make a directory where all temporary files will be kept:

```

19 \makeatletter
20 \ifdefined\eolang@noshell\else\RequirePackage{shellesc}\fi
21 \IfFileExists
22   {\eolang@tmpdir/\jobname}
23   {\message{eolang: Temporary directory "\eolang@tmpdir/\jobname"
24     already exists^^J}}
25   {
26     \ifdefined\eolang@noshell
27       \message{eolang: Temporary directory "\eolang@tmpdir/\jobname"
28         is not created, because of the "noshell" package option,
29         most probably the compilation will fail later^^J}
30     \else
31       \ifnum\ShellEscapeStatus=1
32         \iexec[null]{mkdir -p "\eolang@tmpdir/\jobname"}
33       \else
34         \message{eolang: Temporary directory "\eolang@tmpdir/\jobname"
35           is not created, because -shell-escape is not set, and
36           it doesn't exist, most probably the compilation
37           will fail later^^J}
38       \fi
39     \fi
40   }
41 \makeatother

```

\eolang@lineno Then, we define an internal counter to protect line number from changing:

```

42 \makeatletter\newcounter{eolang@lineno}\makeatother

```

\eolang@mdfive Then, we define a command for MD5 hash calculating of a file:

```

43 \RequirePackage{pdftexcmds}
44 \makeatletter
45 \newcommand\eolang@mdfive[1]{\pdf@filemdfivesum{#1}}
46 \makeatother

```

-phi.pl Then, we create a Perl script for phiquation processing using VerbatimOut environment from [fancyvrb](#):

```

47 \makeatletter
48 \ifdefined\eolang@noshell
49   \message{eolang: Perl script is not going to be created,
50     at "\eolang@tmpdir/\jobname-phi.pl" because of the "noshell"
51     package option^^J}
52 \else
53 \openin 15=\eolang@tmpdir/\jobname-phi.pl
54 \ifeof 15
55 \message{eolang: Perl script is going to be created,
56   because it is absent at "\eolang@tmpdir/\jobname-phi.pl",
57   but if -shell-escape is not set, the compilation will
58   most likely fail now^^J}
59 \begin{VerbatimOut}{\eolang@tmpdir/\jobname-phi.pl}
60 $macro = $ARGV[0];
61 open(my $fh, '<', $ARGV[1]);
62 my $tex; { local $/; $tex = <$fh>; }
63 print "% This file is auto-generated by eolang.sty 0.19.0\n";

```

```

64 print '% There are ', length($tex),
65 ' chars in the input: ', $ARGV[1], "\n";
66 print '% ---', "\n";
67 if (index($tex, "\t") > 0) {
68   print "TABS are prohibited!";
69   exit 1;
70 }
71 my @lines = split (/\\n/g, $tex);
72 foreach my $t (@lines) {
73   print '% ', $t, "\n";
74 }
75 print '% ---', "\n";
76 $tex =~ s/(?<\\)%.*\\n\\n/g;
77 $tex =~ s/^\\s+|\\s+$/g;
78 my $splitting = $tex =~ /^\\begin\\{split\\}/;
79 if ($splitting) {
80   print '% The manual splitting mode is ON since \\begin{split} started the text' . "\n";
81 }
82 my $indents = $tex =~ /\\n +/g;
83 my $gathered = (0 == $indents);
84 if ($gathered) {
85   if ($splitting) {
86     print '% The "gathered" is NOT used because of manual splitting' . "\n";
87     $gathered = 0;
88   } else {
89     print '% The "gathered" is used since all lines are left-aligned' . "\n";
90   }
91 } else {
92   print '% The "gathered" is NOT used because ' .
93     $indents . " lines are indented\n";
94 }
95 my $align = 0;
96 print '% The "align" is NOT used by default' . "\n";
97 if (index($tex, '&&') >= 0) {
98   $macro =~ s/equation/align/g;
99   $align = 1;
100   print '% The "align" is used because of && seen in the text' . "\n";
101 }
102 if ($macro ne 'phiq') {
103   if (not $splitting) {
104     $tex =~ s/\\\\\\n\\n\\n/g;
105     $tex =~ s/\\\\n\\s*/g;
106   }
107   $tex =~ s/\\n*(\\label\\{[\\~\\}+\\})\\n*/\\1/g;
108   $tex =~ s/\\n{3,}/\\n\\n/g;
109 }
110 my @texts = ();
111 sub trep {
112   my ($s) = @_ ;
113   my $open = 0;
114   my $p = 0;
115   for (; $p < length($s); $p++) {
116     $c = substr($s, $p, 1);
117     if ($c eq '}') {

```

```

118     if ($open eq 0) {
119         last;
120     }
121     $open--;
122 }
123 if ($c eq '{') {
124     $open++;
125 }
126 }
127 push(@texts, substr($s, 0, $p));
128 return 'TEXT' . (0+@texts - 1) . '}' . substr($s, $p + 1);
129 }
130 $tex =~ s/\\text\{(.+)/trep("$1")/ge;
131 $tex =~ s/([^\{a-z0-9]|~)QQ(?![a-z0-9])/\\dot{\\Phi}}/g;
132 $tex =~ s/([^\{a-z0-9]|~)Q(?![a-z0-9])/\\Phi}/g;
133 $tex =~ s/([^\{a-z0-9]|~)T(?![a-z0-9])/\\bot}/g;
134 $tex =~ s/([^\{a-z0-9]|~)D>/\\Delta{}/>/g;
135 $tex =~ s/([^\{a-z0-9]|~)L>/\\lambda{}/>/g;
136 $tex =~ s/"([~]+)"/"1"/g;
137 $tex =~ s/(~|(?<=[\s](\\[,.>|/)))([a-zA-Z][a-z0-9]+)(?=[\s](\\[,.>|/)|$)/|2/g;
138 $tex =~ s/([~^]|~)([0-9]+|\\*)\\(\\{[a-z]+|\\{[a-z]+|\\{
139 (->|\\.\\.>|>|:=)/\\alpha_{\\2}\\vert{\\3\\space{\\4}/xg;
140 $tex =~ s/([~^]|~)([0-9]+|\\*)
141 (->|\\.\\.>|>|:=)/\\alpha_{\\2}\\space{\\3}/xg;
142 if ($macro ne 'phiq') {
143     if (not $splitting) {
144         $tex =~ s/\\begin\\{split}\\n\\begin{split}&/g;
145         $tex =~ s/\\n\\s*\\end\\{split}\\n\\end{split}/g;
146         $tex =~ s/\\n\\n\\n\\n\\n/g;
147         $tex =~ s/\\n\\n\\phiEOL{\\n\\n/g;
148         $tex =~ s/\\n\\n\\$/g;
149         $tex =~ s/\\n\\n\\n\\n\\n\\n/g;
150         $tex =~ s/([~&\\s])\\s{2}([~\\s])/1 2/g;
151         $tex =~ s/\\s{2}/ \\quad{/g;
152         $tex = '&' . $tex;
153     }
154     my $lead = '[~\\s]+\\s(?:->|:=|=|==)\\s';
155     my @leads = $tex =~ /&${lead}/g;
156     my @eols = $tex =~ /&/g;
157     if (0+@leads == 0+@eols && 0+@eols > 1) {
158         $tex =~ s/&(${lead})/1~/g;
159         $gathered = 0;
160         print "% The \"gathered\" is NOT used because all ' .
161             (0+@eols) . ' lines are ' . (0+@leads) . \" leads\\n\";
162     }
163 }
164 if ($macro ne 'phiq') {
165     sub strip_tabs {
166         my ($env, $tex) = @_;
167         $tex =~ s/\\//g;
168         return "\\begin{$env}" . $tex . "\\end{$env}";
169     }
170     foreach my $e (('matrix', 'cases')) {
171         $tex =~ s/\\begin\\{\\Q$e\\E*?\\}(\\+\\.\\.\\end\\{\\Q$e\\E*?\\}/strip_tabs($1, $2)/sge;

```

```

172 }
173 }
174 $tex =~ s/\$/\xi{/g;
175 $tex =~ s/(?!\\{)\\^(?!\\{)/\\rho{/g;
176 $tex =~ s/[\\[\\l\\lbracket\\mathbin{}/g;
177 $tex =~ s/[\\]\\mathbin{}/\\rrbracket{/g;
178 $tex =~ s/([\\s,>()]{0-9A-F}{2}(?:-[0-9A-F]{2})+|
179 [0-9]{0-9}(?:\\. [0-9]{0-9}){0-9})(?!\\{)/\\1|\\2|/xg;
180 $tex =~ s/TRUE/|TRUE|/g;
181 $tex =~ s/FALSE/|FALSE|/g;
182 $tex =~ s/\\?/\\varnothing{/g;
183 $tex =~ s/@/\\varphi{/g;
184 $tex =~ s/-([a-z]+)>/\\mathrel{\\phiSlot{\\1}}/g;
185 $tex =~ s/->/\\mathbin{\\mapsto}/g;
186 $tex =~ s/~>/\\mathbin{\\phiWave}/g;
187 $tex =~ s/:=/\\mathrel{\\vDash}/g;
188 $tex =~ s/==/\\mathrel{\\equiv}/g;
189 $tex =~ s/<</\\langle/g;
190 $tex =~ s/>>/\\rangle/g;
191 $tex =~ s/|{2,}/|/g;
192 $tex =~ s/|([^\|]+)|/\\textnormal{\\texttt{\\1}}{/g;
193 $tex =~ s/\\TEXT(\\d+)\\}/'\\text{' . @texts[$1] . '}'/ge;
194 if ($macro eq 'phiq') {
195     print '\\(' if ($tex ne '');
196 } else {
197     print '\\begin{' , $macro , "\\n";
198     if (not($align)) {
199         if ($gathered) {
200             print '\\begin{gathered}' . "\\n";
201         } elsif (not $splitting) {
202             print '\\begin{split}' . "\\n";
203         }
204     }
205 }
206 if ($gathered and not($align)) {
207     $tex =~ s/^&/g;
208     $tex =~ s/\\n&/\\n/g;
209 }
210 print $tex;
211 if ($macro eq 'phiq') {
212     print '\\)' if ($tex ne '');
213 } else {
214     if (not($align)) {
215         if ($gathered) {
216             print "\\n" . '\\end{gathered}';
217         } elsif (not $splitting) {
218             print "\\n" . '\\end{split}';
219         }
220     }
221     print "\\n" . '\\end{' . $macro . '}' ;
222 }
223 print '\\endinput';
224 \\end{VerbatimOut}
225 \\message{eolang: File with Perl script

```

```

226 '\eolang@tmpdir/\jobname-phi.pl' saved^^J}
227 \else
228 \message{eolang: Perl script already exists at
229 "\eolang@tmpdir/\jobname-phi.pl"^^J}
230 \fi
231 \closein 15
232 \fi
233 \makeatother

```

`\phiSaveTo` Then, we define the `\phiSaveTo` command to instruct the `phi`uation environment that the output should not be sent to the document but saved to the file instead:

```

234 \makeatletter
235 \newcommand\phiSaveTo[1]{\def\eolang@phiSaveTo{#1}}
236 \makeatother

```

`\eolang@tmp` Then, we define the `\eolang@tmp` command, which generates temporary file names:

```

237 \makeatletter
238 \newcommand\eolang@tmp[1]{#1\ifxetex-xe\else\ifluatex-lua\fi\fi.tex}
239 \makeatother

```

`\eolang@ifabsent` Then, we define the `\eolang@ifabsent` command, which if a given file is absent, runs a processing command, otherwise just inputs it:

```

240 \makeatletter
241 \newcommand\eolang@ifabsent[2]{%
242 \IfFileExists
243 {#1}
244 {%
245 \message{eolang: File "#1" already exists ^^J}%
246 \input{#1}}
247 {%
248 \ifdefined\eolang@noshell%
249 \message{eolang: Shell processing is disabled^^J}%
250 \else%
251 \ifnum\ShellEscapeStatus=1\else%
252 \message{eolang: The -shell-escape command line
253 option is not provided, most probably compilation
254 will fail now:^^J}%
255 \fi%
256 #2%
257 \fi%
258 }%
259 }
260 \makeatother

```

`phi`uation Then, we define the `phi`uation and the `phi`uation* environments through a supplementary `\eolang@process` command:

```

261 \makeatletter\newcommand\eolang@process[1]{
262 \def\hash{\eolang@mdfive
263 {\eolang@tmpdir/\jobname/\eolang@tmp{phiuation}}-\the\inputlineno}%
264 \eolang@ifabsent
265 {\eolang@tmpdir/\jobname/\eolang@tmp{\hash-phiuation-post}}
266 {%
267 \iexec[null]{cp "\eolang@tmpdir/\jobname/\eolang@tmp{phiuation}"

```

```

268     "\eolang@tmpdir/\jobname/\eolang@tmp{\hash-phiuation}"}%
269     \message{Start parsing 'phi' at line no. \the\inputlineno^^J}
270     \iexec[trace,stdout=\eolang@tmpdir/\jobname/\eolang@tmp{\hash-phiuation-post}]{
271         perl "\eolang@tmpdir/\jobname-phi.pl"
272         '#1'
273         "\eolang@tmpdir/\jobname/\eolang@tmp{\hash-phiuation}"
274         \ifdefined\eolang@nocomments | perl -pe 's/\%.*(\n|$)//g'\fi
275         \ifdefined\eolang@phiSaveTo > \eolang@phiSaveTo\fi}%
276     }%
277     \setcounter{FancyVerbLine}{\value{eolang@lineno}}%
278     \def\eolang@phiSaveTo{\relax}%
279 }
280 %
281 \newenvironment{phiuation*}%
282 {\catcode'\|=12 \VerbatimEnvironment%
283 \setcounter{eolang@lineno}{\value{FancyVerbLine}}%
284 \begin{VerbatimOut}
285   {\eolang@tmpdir/\jobname/\eolang@tmp{phiuation}}
286 {\end{VerbatimOut}\eolang@process{equation*}}
287 %
288 \newenvironment{phiuation}%
289 {\catcode'\|=12 \VerbatimEnvironment%
290 \setcounter{eolang@lineno}{\value{FancyVerbLine}}%
291 \begin{VerbatimOut}
292   {\eolang@tmpdir/\jobname/\eolang@tmp{phiuation}}
293 {\end{VerbatimOut}\eolang@process{equation*}}
294 \makeatother

```

\phiq Then, we define \phiq command:

```

295 \RequirePackage{xstring}
296 \makeatletter\newcommand\phiq[1]{%
297   \StrSubstitute{\detokenize{#1}}{'}{'"''}[\clean]%
298   \def\hash{\pdf@mdfivesum{\clean}-\the\inputlineno}%
299   \ifdefined\eolang@nodollar\else\catcode'\$=3 \fi%
300   \eolang@ifabsent
301     {\eolang@tmpdir/\jobname/\eolang@tmp{\hash-phiq-post}}
302     {%
303       \iexec[log,trace,quiet,stdout=\eolang@tmpdir/\jobname/\eolang@tmp{phiq}]{
304         printf '\%s' '\clean'}%
305       \iexec[quiet,null]{cp "\eolang@tmpdir/\jobname/\eolang@tmp{phiq}"
306         "\eolang@tmpdir/\jobname/\eolang@tmp{\hash-phiq}"}%
307       \iexec[trace,stdout=\eolang@tmpdir/\jobname/\eolang@tmp{\hash-phiq-post}]{
308         perl \eolang@tmpdir/\jobname-phi.pl 'phiq'
309         "\eolang@tmpdir/\jobname/\eolang@tmp{\hash-phiq}"
310         \ifdefined\eolang@nocomments | perl -pe 's/\%.*(\n|$)//g' \fi}%
311       \message{eolang: Parsed 'phiq' at line no. \the\inputlineno^^J}%
312     }%
313   \ifdefined\eolang@nodollar\else\catcode'\$=\active\fi%
314 }\makeatother

```

nodollar Then, we redefine dollar sign:

```

315 \ifdefined\eolang@nodollar\else
316   \begingroup
317   \catcode'\$=\active

```

```

318 \protected\gdef$#1${\phi{#1}}
319 \endgroup
320 \AtBeginDocument{\catcode'\$=\active}
321 \fi

```

-sodg.pl Then, we create a Perl script for sodg graphs processing using VerbatimOut from [fancyvrb](#):

```

322 \makeatletter
323 \ifdefined\eolang@noshell
324 \message{eolang: Perl script is not going to be created
325   at "\eolang@tmpdir/\jobname-sodg.pl", because of the
326   "noshell" package option^^J}
327 \else
328 \openin 15=\eolang@tmpdir/\jobname-sodg.pl
329 \ifeof 15
330 \message{eolang: Perl script is going to be created,
331   because it is absent at "\eolang@tmpdir/\jobname-sodg.pl",
332   but if -shell-escape is not set, the compilation will
333   most likely fail now^^J}
334 \begin{VerbatimOut}{\eolang@tmpdir/\jobname-sodg.pl}
335 sub num {
336   my ($i) = @_;
337   $i =~ s/(\+|-)\./\10./g;
338   return $i;
339 }
340 sub fmt {
341   my ($tex) = @_;
342   $tex =~ s/|([^\|]+)|\\|\\textnormal{\\texttt{\\1}}/g;
343   return $tex;
344 }
345 sub toem {
346   my ($cm) = @_;
347   return $cm * 2.8;
348 }
349 sub vertex {
350   my ($v) = @_;
351   if (index($v, 'v0') == 0) {
352     return '\Phi';
353   } else {
354     $v =~ s/^v/v_/g;
355     $v =~ s/^0-9]$/g;
356     return $v . '>';
357   }
358 }
359 sub tailor {
360   my ($t, $m) = @_;
361   $t =~ s/<([A-Z]?[A-Z]?):([>]+)>/\2/g;
362   $t =~ s/<[A-Z]+:([>]+)>/\2/g;
363   return $t;
364 }
365 open(my $fh, '<', $ARGV[0]);
366 my $tex; { local $/; $tex = <$fh>; }
367 if (index($tex, "\t") > 0) {
368   print "TABS are prohibited!";

```

```

369 exit 1;
370 }
371 print '% This file is auto-generated', "\n%\n";
372 print '% --- there are ', length($tex),
373 ' chars in the input (', $ARGV[0], "):\n";
374 foreach my $t (split (/\\n/g, $tex)) {
375   print '% ', $t, "\n";
376 }
377 print "% ---\n";
378 $tex =~ s/\\\\/\\n/g;
379 $tex =~ s/\\n/\\n/g;
380 $tex =~ s/(\\[a-zA-Z]+)\\s+/\\1/g;
381 $tex =~ s/\\n{2,}/\\n/g;
382 my @cmds = split(/\\n/g, $tex);
383 print '% --- before processing:' . "\n";
384 foreach my $t (split (/\\n/g, $tex)) {
385   print '% ', $t, "\n";
386 }
387 print '% ---';
388 print ' (' . (0+@cmds) . " lines)\n";
389 print '\\begin{picture}', "\n";
390 for (my $c = 0; $c < 0+@cmds; $c++) {
391   my $cmd = $cmds[$c];
392   $cmd =~ s/^\\s+//g;
393   $cmd =~ s/(?!\\)%.*/g;
394   my ($head, $tail) = split(/ /, $cmd, 2);
395   my %opts = {};
396   my ($body, $style) = split(/style:/, $tail, 2);
397   $opts{'style'} = $style;
398   $tail = $body;
399   foreach my $p (split(/ /, $tail)) {
400     my ($q, $t) = split(/:/, $p);
401     $opts{$q} = $t;
402   }
403   if (index($head, '\\') == 0) {
404     print $cmd;
405   } elsif (index($head, '->') >= 0) {
406     my $draw = '\\draw[';
407     if (exists $opts{'pi'}) {
408       $draw = $draw . '<MB:phi-pi><F:draw=none>';
409       if (not exists $opts{'a'}) {
410         $opts{'a'} = '\\pi';
411       }
412     }
413     if (exists $opts{'rho'} and not(exists $opts{'bend'})) {
414       $draw = $draw . '<MB:,phi-rho>';
415     }
416     $draw = $draw . ',' . $opts{'style'} . ']';
417     my ($from, $to) = split (/>/, $head);
418     $draw = $draw . " ($from) ";
419     if (exists $opts{'bend'}) {
420       $draw = $draw . 'edge [<F:draw=none><MF:,bend right=' .
421         num($opts{'bend'}) . '>';
422       if (exists $opts{'rho'}) {

```

```

423     $draw = $draw . '<MB:,phi-rho>';
424 }
425 $draw = $draw . ']'';
426 } else {
427     $draw = $draw . '--';
428 }
429 if (exists $opts{'a'}) {
430     my $a = $opts{'a'};
431     if (index($a, '$') == -1) {
432         $a = '$' . fmt($a) . '$';
433     } else {
434         $a = fmt($a);
435     }
436     $draw = $draw . '<MB: node [phi-attr] {' . $a . '}>';
437 }
438 if (exists $opts{'break'}) {
439     $draw = $draw . '<F: coordinate [pos=' .
440         ($opts{'break'} / 100) . ']' (break)>';
441 }
442 $draw = $draw . " (<MF:${to}><B:break-v>)"';
443 if (exists $opts{'break'}) {
444     print tailor($draw, 'F') . ";\n";
445     print ' \node[outer sep=' . toem(0.1) . 'em,inner sep=0em] ' .
446         'at (break) (break-v) {' . vertex($to) .
447         '$};' . "\n";
448     print ' ' . tailor($draw, 'B');
449 } else {
450     print tailor($draw, 'M');
451 }
452 } elsif (index($head, '=>') >= 0) {
453     my ($from, $to) = split (/=>/, $head);
454     my $size = () = $head =~ /=/g;
455     if ($from eq '') {
456         print '\node [phi-arrow, left=' . toem($size * 0.6) . 'em of ' .
457             $to . '.center]';
458     } elsif ($to eq '') {
459         print '\node [phi-arrow, right=' . toem($size * 0.6) . 'em of ' .
460             $from . '.center]';
461     } else {
462         print '\node [phi-arrow] at ($(' .
463             $from . ')!0.5!(' . $to . ')$)';
464     }
465     print '{}';
466 } elsif (index($head, '!!') >= 0) {
467     my ($v, $marker) = split (/!+/, $head);
468     my $size = () = $head =~ /*!/g;
469     print '\node [phi-marker, left=' .
470         toem($size * 0.6) . 'em of ' .
471         $v . '.center'][' . fmt($marker) . ']'';
472 } elsif (index($head, '++') >= 0) {
473     my ($v, $suffix) = split (/+/, $head);
474     my @friends = ($v);
475     foreach my $c (@cmds) {
476         $e = $c;

```

```

477     $e =~ s/^\s+//g;
478     my $h = $e;
479     $h = substr($e, 0, index($e, ' ')) if index($e, ' ') >= 0;
480     foreach my $f (@friends) {
481         my $add = '';
482         if (index($h, $f . '->') >= 0) {
483             $add = substr($h, index($h, '->') + 2);
484         }
485         if ($h =~ /->\Q${f}\E$/) {
486             $add = substr($h, 0, index($h, '->'));
487         }
488         if (index($e, ' xy:' . $f . ',') >= 0) {
489             $add = $h;
490         }
491         if (index($add, '+') == -1
492             and $add ne ''
493             and not(grep(/^Q${add}\E$/, @friends))) {
494             push(@friends, $add);
495         }
496     }
497 }
498 my @extra = ();
499 foreach my $e (@cmds) {
500     $m = $e;
501     if ($m =~ /\s*\Q${v}\E\s/) {
502         next;
503     }
504     if ($m =~ /\s*[^\s]+\s/ and not($m =~ /\s*\Q${head}\E\s/)) {
505         next;
506     }
507     foreach my $f (@friends) {
508         my $h = $f;
509         $h =~ s/[a-z]$/g;
510         if ($m =~ s/^(s*)\Q${f}\E\+\Q${suffix}\E\s?/\1${h}${suffix} /g) {
511             last;
512         }
513         $m =~ s/^(s*)\Q${f}\E\s/\1${h}${suffix} /g;
514         $m =~ s/^(s*)\Q${f}\E->/\1${h}${suffix}->/g;
515         $m =~ s/\sxy:\Q${f}\E,/ xy:${h}${suffix},/g;
516         $m =~ s/->\Q${f}\E\s/->${h}${suffix} /g;
517     }
518     if ($m ne $e) {
519         push(@extra, ' ' . $m);
520     }
521 }
522 splice(@extra, 0, 0, @extra[-1]);
523 splice(@extra, -1, 1);
524 splice(@extra, 0, 0, '% clone of ' . $v . ' (' . $head .
525     '), friends: [' . join(', ', @friends) . ']' in ' .
526     (0+@cmds) . ' lines');
527 splice(@cmds, $c, 1, @extra);
528 print '% cloned ' . $v . ' at line no.' . $c .
529     ' (+ ' . (0+@extra) . ' lines -> ' .
530     (0+@cmds) . ' lines total)';

```

```

531 } elseif ($head =~ /^v[0-9]+[a-z]?$/ ) {
532   print '\node[';
533   if (exists $opts{'xy'}) {
534     my ($v, $right, $down) = split(/,/ , $opts{'xy'});
535     my $loc = '';
536     if ($down > 0) {
537       $loc = 'below';
538     } elseif ($down < 0) {
539       $loc = 'above';
540     }
541     if ($right > 0) {
542       $loc = $loc . 'right';
543     } elseif ($right < 0) {
544       $loc = $loc . 'left';
545     }
546     print ', ' . $loc . '=';
547     print toem(abs(num($down))) . 'em and ' .
548       toem(abs(num($right))) . 'em of ' . $v . '.center';
549   }
550   if (exists $opts{'data'}) {
551     print ',phi-data';
552     if ($opts{'data'} ne '') {
553       my $d = $opts{'data'};
554       if (index($d, '|') == -1) {
555         $d = '$\Delta\phiDotted\text{' .
556           '\textnormal{\texttt{' . fmt($d) . '}}}$';
557       } else {
558         $d = fmt($d);
559       }
560       $opts{'box'} = $d;
561     }
562   } elseif (exists $opts{'atom'}) {
563     print ',phi-atom';
564     if ($opts{'atom'} ne '') {
565       my $a = $opts{'atom'};
566       if (index($a, '$') == -1) {
567         $a = '$\lambda\phiDotted{' . fmt($a) . '$';
568       } else {
569         $a = fmt($a);
570       }
571       $opts{'box'} = $a;
572     }
573   } else {
574     print ',phi-object';
575   }
576   if (exists $opts{'edgeless'}) {
577     print ',draw=none';
578   }
579   print ', ' . $opts{'style'} . ']';
580   print ' (' . $head . ')';
581   print '{';
582   if (exists $opts{'tag'}) {
583     my $t = $opts{'tag'};
584     if (index($t, '$') == -1) {

```

```

585     $t = '$' . $t . '$';
586   } else {
587     $t = fmt($t);
588   }
589   print $t;
590 } else {
591   print '$' . vertex($head) . '$';
592 }
593 print '>';
594 if (exists $opts{'box'}) {
595   print ' node[phi-box] at (';
596   print $head, '.south east) {';
597   print $opts{'box'}, '>';
598 }
599 }
600 print ";\n";
601 }
602 print '\end{phicture}%', "\n";
603 print "% --- after processing:\n%";
604 foreach my $c (@cmds) {
605   print '% ', $c, "\n";
606 }
607 print '% --- (' . (0+@cmds) . " lines)\n";
608 print '\endinput';
609 \end{VerbatimOut}
610 \message{eolang: File with Perl script
611   '\eolang@tmpdir/\jobname-sodg.pl' saved^^J}
612 \else
613   \message{eolang: Perl script already exists at
614     "\eolang@tmpdir/\jobname-sodg.pl"^^J}
615 \fi
616 \closein 15
617 \fi
618 \makeatother

```

FancyVerbLine Then, we reset the counter for [fancyvrb](#), so that it starts counting lines from zero when the document starts rendering:

```

619 \setcounter{FancyVerbLine}{0}

```

tikz Then, we include [tikz](#) package and its libraries:

```

620 \RequirePackage{tikz}
621 \usetikzlibrary{arrows}
622 \usetikzlibrary{shapes}
623 \usetikzlibrary{decorations}
624 \usetikzlibrary{decorations.pathmorphing}
625 \usetikzlibrary{decorations.pathreplacing}
626 \usetikzlibrary{positioning}
627 \usetikzlibrary{calc}
628 \usetikzlibrary{math}
629 \usetikzlibrary{arrows.meta}

```

phicture Then, we define internal environment `phicture`:

```

630 \newenvironment{phicture}%
631 {\noindent\begin{tikzpicture}[

```

```

632 ->,>=stealth',node distance=0,line width=.08em,
633 pics/parallel arrow/.style={
634   code={\draw[-latex,phi-rho] (##1) -- (-##1);}}}%
635 {\end{tikzpicture}}
636 \tikzstyle{phi-arrow} = [fill=white!80!black, single arrow,
637   minimum height=0.05em, minimum width=0.05em,
638   single arrow head extend=2mm]
639 \tikzstyle{phi-marker} = [inner sep=0pt, minimum height=1.4em,
640   minimum width=1.4em, font={\small\color{white}\ttfamily},
641   fill=gray]
642 \tikzstyle{phi-thing} = [inner sep=0pt,minimum height=2.4em,
643   draw,font={\small}]
644 \tikzstyle{phi-object} = [phi-thing,circle]
645 \tikzstyle{phi-data} = [phi-thing,regular polygon,
646   regular polygon sides=8]
647 \tikzstyle{phi-empty} = [phi-object]
648 \tikzset{%
649   phi-rho/.style={
650     postaction={%
651       decoration={
652         show path construction,
653         curveto code={
654           \tikzmath{
655             coordinate \I, \F, \v;
656             \I = (\tikzinputsegmentfirst);
657             \F = (\tikzinputsegmentlast);
658             \v = ($(\I) -(\F)$);
659             real \d, \a, \r, \t;
660             \d = 0.8;
661             \t = atan2(\vy, \vx);
662             if \vx<0 then { \a = 90; } else { \a = -90; };
663             {
664               \draw[arrows={-latex}, decorate,
665                 decoration={%
666                   snake, amplitude=.4mm,
667                   segment length=2mm,
668                   post length=1mm
669                 }]
670               ($(\F)!.5!(\I) +(\t: -\d em) +(\t +\a: 1ex)$)
671               -- ++(\t: 2*\d em);
672             };
673           }
674         },
675     lineto code={
676       \tikzmath{
677         coordinate \I, \F, \v;
678         \I = (\tikzinputsegmentfirst);
679         \F = (\tikzinputsegmentlast);
680         \v = ($(\I) -(\F)$);
681         real \d, \a, \r, \t;
682         \d = 0.8;
683         \t = atan2(\vy, \vx);
684         if \vx<0 then { \a = 90; } else { \a = -90; };
685         {

```

```

686         \draw[arrows={-latex}, decorate,
687         decoration={%
688             snake, amplitude=.4mm,
689             segment length=2mm,
690             post length=1mm}]
691         ($(\F)!.5!(\I) +(\t: -\d em) +(\t +\a: 1ex)$)
692         -- ++(\t: 2*\d em);
693     };
694 }
695 }
696 },
697 decorate
698 }
699 }
700 }
701 \tikzstyle{phi-pi} = [draw,dotted]
702 \tikzstyle{phi-atom} = [phi-object,double]
703 \tikzstyle{phi-box} = [xshift=-5pt,yshift=3pt,draw,fill=white,
704     rectangle,line width=.04em,minimum width=1.2em,anchor=north west,
705     font={\scriptsize}]
706 \tikzstyle{phi-attr} = [midway,sloped,inner sep=0pt,
707     above=2pt,sloped/.append style={transform shape},
708     font={\scriptsize},color=black]

```

\sodgSaveTo Then, we define the \sodgSaveTo command to instruct the sodg environment that the output should not be sent to the document but saved to the file instead:

```

709 \makeatletter
710 \newcommand\sodgSaveTo[1]{\def\eolang@sodgSaveTo{#1}}
711 \makeatother

```

sodg Then, we create a new environment sodg, as suggested [here](#):

```

712 \makeatletter\newenvironment{sodg}%
713 {\catcode'\|=12 \VerbatimEnvironment%
714 \setcounter{eolang@lineno}{\value{FancyVerbLine}}%
715 \begin{VerbatimOut}
716   {\eolang@tmpdir/\jobname/\eolang@tmp{sodg}}
717 {\end{VerbatimOut}}%
718 \def\hash{\eolang@mdfive
719   {\eolang@tmpdir/\jobname/\eolang@tmp{sodg}}-\the\inputlineno}%
720 \catcode'\$=3 %
721 \eolang@ifabsent
722   {\eolang@tmpdir/\jobname/\eolang@tmp{\hash-sodg-post}}
723   {%
724     \iexec[null]{cp "\eolang@tmpdir/\jobname/\eolang@tmp{sodg}"
725       "\eolang@tmpdir/\jobname/\eolang@tmp{\hash-sodg}}}%
726     \message{eolang: Start parsing 'sodg' at line no. \the\inputlineno^^J}
727     \iexec[trace,stdout=\eolang@tmpdir/\jobname/\eolang@tmp{\hash-sodg-post}]{
728       perl "\eolang@tmpdir/\jobname-sodg.pl"
729       "\eolang@tmpdir/\jobname/\eolang@tmp{\hash-sodg}"
730       \ifdefined\eolang@nocomments | perl -pe 's/\%.*(\n|$)//g'\fi
731       \ifdefined\eolang@sodgSaveTo > \eolang@sodgSaveTo\fi}%
732   }
733 \catcode'\$=active%
734 \setcounter{FancyVerbLine}{\value{eolang@lineno}}%

```

```

735 \def\eolang@sodgSaveTo{\relax}%
736 }\makeatother

```

`\eoAnon` Then, we define a supplementary command to help us anonymize some content.

```

737 \RequirePackage{hyperref}
738 \pdfstringdefDisableCommands{
739 \def\({}%
740 \def\)}%
741 \def\alpha{alpha}%
742 \def\varphi{phi}%
743 }
744 \makeatletter
745 \NewExpandableDocumentCommand{\eoAnon}{O{ANONYMIZED}m}{%
746 \ifdefined\eolang@anonymous%
747 \textcolor{orange}{#1}%
748 \else%
749 #2%
750 \fi%
751 }\makeatother

```

`\eolang` Then, we define a simple supplementary command to help you print EO, the name of our language.

```

752 \newcommand\eolang{%
753 \eoAnon[XYZ]{\sffamily EO}}

```

`\phic` Then, we define a simple supplementary command to help you print φ -calculus, the name of our formal apparatus.

```

754 \newcommand\phic{%
755 \eoAnon[(\alpha)-cal-cu-lus]{(\varphi)-cal-cu-lus}}

```

`\xmirl` Then, we define a simple supplementary command to help you print XMIR, the name of our XML-based format of program representation.

```

756 \newcommand\xmirl{%
757 \eoAnon[XML{^+}]{XMIR}}

```

`\phiConst` Then, we define a command to render an arrow for a constant attribute, as suggested [here](#):

```

758 \newcommand\phiConst{%
759 \mathrel{\hspace{.15em}}%
760 \mapstochar\mathrel{\hspace{-.15em}}\mapsto}

```

`\phiWave` Then, we define a command to render an arrow for a multi-layer attribute, as suggested [here](#):

```

761 \newcommand\phiWave{%
762 \mapstochar\mathrel{\mspace{0.45mu}}\leadsto}

```

`\phiSlot` Then, we define a command to render an arrow for a slot in a basket:

```

763 \newcommand\phiSlot[1]{%
764 \xrightarrow{\text{\sffamily\scshape #1}}}

```

`\phi0set` Then, we define two commands to position a text over and under an arrow, as suggested [here](#):

```

765 \makeatletter
766 \newcommand{\phiOset}[2]{%
767   \mathrel{\mathop{\#2}\limits^{
768     \vbox to 0ex{\kern-2\ex@
769       \hbox{$\scriptscriptstyle\#1$\vss}}}}
770 \newcommand{\phiUset}[2]{%
771   \mathrel{\mathop{\#2}\limits_{
772     \vbox to 0ex{\kern-6.3\ex@
773       \hbox{$\scriptscriptstyle\#1$\vss}}}}
774 \makeatother

```

`\phiMany` Then, we define a command for an arrow with iterating indices:

```

775 \newcommand{\phiMany}[3]{%
776   \phiOset{\#3}{\phiUset{\#2}{\#1}}

```

`\phiEOL` Then, we define a command for line breaks in formulas:

```

777 \newcommand{\phiEOL}{\[-4pt]}

```

`\phiDotted` Then, we define a command to render an arrow for a special attribute, as suggested [here](#):

```

778 \RequirePackage{trimclip}
779 \RequirePackage{amsfonts}
780 \makeatletter
781 \newcommand{\phiDotted}{%
782   \mapstochar\mathrel{\mathpalette\phiDotted@\relax}}
783 \newcommand{\phiDotted@}[2]{%
784   \begin{group}%
785     \settowidth{\dimen\z@}{\m@th\#1\rightarrow}%
786     \settoheight{\dimen\tw@}{\m@th\#1\rightarrow}%
787     \sbox\z@{%
788       \makebox[\dimen\z@][s]{%
789         \clipbox{0 0 {0.4\width} 0}%
790         {\resizebox{\dimen\z@}{\height}%
791           {\m@th\#1\dashrightarrow}}}%
792       \hss%
793       \clipbox{{0.69\width} {-0.1\height} 0
794         {-\height}}{\m@th\#1\rightarrow}%
795     }%
796   }%
797   \ht\z@=\dimen\tw@ \dp\z@=\z@%
798   \box\z@%
799   \end{group}%
800 }
801 \makeatother

```