



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

Yegor Bugayenko
yegor256@gmail.com

2025/11/24, 0.20.2

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](#)

<pre> app ↦ [ρ ↦ ξ.b.², 0/t ↦ TRUE, b ↦ [α* ↦ Φ.fn(56), φ ↦ Φ.string.trim(ξ), Δ ↦ 01-FE-C3]], x ↦ [λ ↦ ∅]. </pre>	<pre> 1 \documentclass{minimal} 2 \usepackage{eolang} 3 \begin{document} 4 \begin{phiqutation*} 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{phiqutation*} 12 \end{document} </pre>
---	--

`phiqutation (env)` The environment `phiqutation` 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 “ \mapsto ” (`\phiDotted`),
- “~>” maps to “ \rightsquigarrow ” (`\phiWave`),
- “D>” maps to “ $\Delta \mapsto$ ” (`\Delta .>`),
- “L>” maps to “ $\lambda \mapsto$ ” (`\lambda .>`),
- “[[” maps to “[” (`\llbracket`),
- “]]” maps to “]” (`\rrbracket`),
- “==” maps to “ \equiv ” (`\equiv`),
- “|abc|” maps to “`abc`” (`\texttt{abc}`).

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

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

Before any arrow you can put a number, which will be rendered as `\alpha` with an index, for example `\phiiq{0->x}` will render “ $\alpha_0 \mapsto x$ ”. It’s also possible to use `~0` to render α_0 . 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.

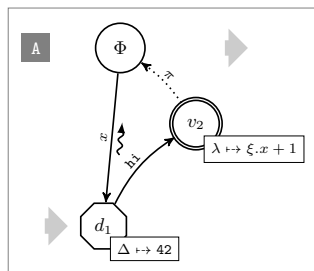
Object names are automatically converted to fixed-width font too, if they have more than one letter.

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:

<p>A simple object $x \mapsto [\varphi \mapsto y]$ is a decorator of the data object $y \mapsto [\Delta \mapsto 42]$.</p>	<pre> 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} </pre>
---	--

`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
`\phic` language. It understands the anonymous package option and prints itself differently, to
`\xmir` double-blind your paper. There is also `\phic` command to print the name of φ -calculus,
also sensitive to anonymous mode. The macro `\xmir` prints “XMIR”.

<p>In our research we use XYZ, an experimental object-oriented dataflow language, α-calculus, as its formal foundation, and XML⁺ — its XML-based representation.</p>	<pre> 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 \xmir{} --- \\ 9 its XML-based representation. 10 \end{document} </pre>
---	--

Without the anonymous option there will be no orange color:

<p>In our research we use EO, an experimental object-oriented dataflow language, φ-calculus, as its formal foundation, and XMIR — its XML-based representation.</p>	<pre> 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 \xmir{} --- \\ 9 its XML-based representation. 10 \end{document} </pre>
--	---

`\phiWave`
`\phiDotted`

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, while $x \rightarrow y$ makes it a special attribute.

```

6 If $$ 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 while $x \phiDotted y$ makes it
11 a special attribute.
12 \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:

$\left[\rightarrow^* \right]$.

```

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 $[1 \rightarrow x_1, 2 \rightarrow x_2, \dots, \alpha_n \rightarrow x_n]$ and expression $[\alpha_i \xrightarrow{i} x_i]$ 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: $\left[x \mapsto \emptyset \right]$
Bound: $\left[x \mapsto \left[\Delta \mapsto 42 \right] \right]$

```

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

$\frac{x \mapsto [\varphi \mapsto y] \quad y \mapsto [z \mapsto 42]}{x.z \mapsto \perp} R1$	<pre>6 \begin{phiquation*} 7 \dfrac \ 8 {x->[[@->y]] \quad y->[[z->42]]} \ 9 {x.z -> T} \ 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}`:

$\frac{x \mapsto [\varphi \mapsto y, z \mapsto 42, \quad 0/g \mapsto \emptyset, 1/foo \mapsto 42]}{x \mapsto [\varphi \mapsto y, z \mapsto \emptyset, f \rightsquigarrow \text{pi}(\quad 0 \mapsto [\psi \mapsto \text{hello}(12)], \quad \alpha_1 \mapsto 42)]} R2.$	<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)]] 14 \end{split}}\text{R2}. 15 \end{phiquation*}</pre>
---	---

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

$\text{foo} \mapsto \left\{ \begin{array}{c} \emptyset \\ [\lambda \mapsto \rho \times \xi \cdot \alpha_0] \\ [\Delta \mapsto 42] \end{array} \right\}$	<pre> 5 \begin{phiqutation*} 6 foo -> \left{\begin{matrix} \backslash 7 ? \backslash 8 [[L> ~ \times \$.\alpha_0]] \backslash 9 [[D> 42]] \backslash 10 \end{matrix}\right\} 11 \end{phiqutation*} </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{phiqutation*} 6 \beta := \begin{cases} \backslash 7 [v_2, @ -dtzd> 42] \backslash 8 [v_{33}] \backslash 9 \end{cases} 10 \end{phiqutation*} 11 \end{document} </pre>
--	--

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

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

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

<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>	<pre> 6 Discriminant can be calculated using 7 the following simple formula: 8 \begin{phiqutation} 9 D = b^{^2} - {4}ac. 10 \label{d} 11 \end{phiqutation} 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):

$\llbracket 0 \mapsto x \rrbracket$	This is formation	6	<code>\begin{phiquestion*}</code>
$\llbracket 0 \mapsto \emptyset \rrbracket$	Abstraction	7	<code>[[0->x]] && \text{This is formation}</code>
$x(\Delta \mapsto 42)$	Application	8	<code>[[0->?]] && \text{Abstraction}</code>
		9	<code>x(D>42) && \text{Application}</code>
		10	<code>\end{phiquestion*}</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	The object formation $\$[\llbracket 0 \mapsto x \rrbracket]\$$
	7	may be replaced with a formula
	8	<code>\(Q \times a^2 \)</code> .

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

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

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

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

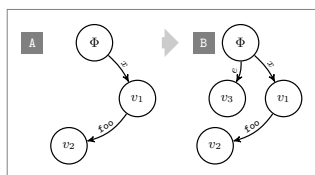
It is possible to use “manual splitting” mode in the `phiquestion` environment by starting the body with `\begin{split}`:

$ \begin{aligned} x(\pi) &\mapsto 4 \\ x(a, b, c) &\mapsto \llbracket \alpha_0 \mapsto \emptyset \rrbracket \end{aligned} $	5	<code>\begin{phiquestion*}</code>
	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{phiquestion*}</code>

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

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

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

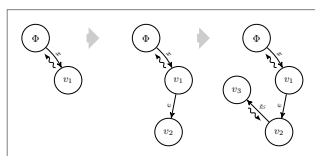


```

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:

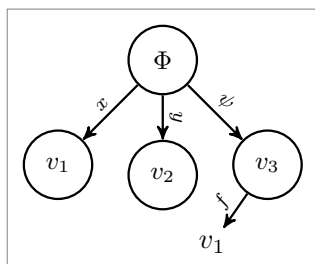


```

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:

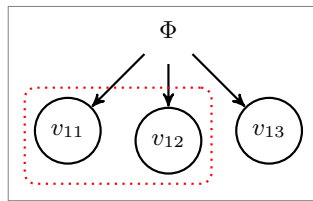


```

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:

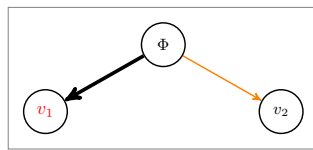


```

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 nodollar/.store in=\eolang@nodollar,
15 anonymous/.store in=\eolang@anonymous,
16 noshell/.store in=\eolang@noshell,
17 tmpdir

```

```

18 }
19 \ProcessPgfPackageOptions{/eolang}
    Then, we make a directory where all temporary files will be kept:
20 \makeatletter
21 \ifdefined\eolang@noshell\else\RequirePackage{shellesc}\fi
22 \IfFileExists
23   {\eolang@tmpdir/\jobname}
24   {\message{eolang: Temporary directory "\eolang@tmpdir/\jobname"
25     already exists^^J}}
26   {
27     \ifdefined\eolang@noshell
28       \message{eolang: Temporary directory "\eolang@tmpdir/\jobname"
29         is not created, because of the "noshell" package option,
30         most probably the compilation will fail later^^J}
31     \else
32       \ifnum\ShellEscapeStatus=1
33         \iexec[null]{mkdir -p "\eolang@tmpdir/\jobname"}
34       \else
35         \message{eolang: Temporary directory "\eolang@tmpdir/\jobname"
36           is not created, because -shell-escape is not set, and
37           it doesn't exist, most probably the compilation
38           will fail later^^J}
39       \fi
40     \fi
41   }
42 \makeatother

```

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

```
43 \makeatletter\newcounter{eolang@lineno}\makeatother
```

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

```

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

```

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

```

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

```

```

63 my $tex; { local $/; $tex = <$fh>; }
64 print "% This file is auto-generated by eolang.sty 0.20.2\n";
65 print '% There are ', length($tex),
66 ' chars in the input: ', $ARGV[1], "\n";
67 print '% ---', "\n";
68 if (index($tex, "\t") > 0) {
69   print "TABS are prohibited!";
70   exit 1;
71 }
72 my @lines = split (/\\n/g, $tex);
73 foreach my $t (@lines) {
74   print '% ', $t, "\n";
75 }
76 print '% ---', "\n";
77 $tex =~ s/(?<!\\)%.*\\n\\n/g;
78 $tex =~ s/^\\s+|\\s+$//g;
79 my $splitting = $tex =~ /^\\begin\\{split\\}/;
80 if ($splitting) {
81   print '% The manual splitting mode is ON since \\begin{split} started the text' . "\n";
82 }
83 my $indents = $tex =~ /\\n +/g;
84 my $gathered = (0 == $indents);
85 if ($gathered) {
86   if ($splitting) {
87     print '% The "gathered" is NOT used because of manual splitting' . "\n";
88     $gathered = 0;
89   } else {
90     print '% The "gathered" is used since all lines are left-aligned' . "\n";
91   }
92 } else {
93   print '% The "gathered" is NOT used because ' .
94     $indents . " lines are indented\n";
95 }
96 my $align = 0;
97 print '% The "align" is NOT used by default' . "\n";
98 if (index($tex, '&&') >= 0) {
99   $macro =~ s/equation/align/g;
100   $align = 1;
101   print '% The "align" is used because of && seen in the text' . "\n";
102 }
103 if ($macro ne 'phiq') {
104   if (not $splitting) {
105     $tex =~ s/\\\\\\n\\n\\n/g;
106     $tex =~ s/\\\\\\n\\s*//g;
107   }
108   $tex =~ s/\\n*(\\label\\{[~\\}+\\})\\n*/\\1/g;
109   $tex =~ s/\\n{3,}/\\n\\n/g;
110 }
111 my @texts = ();
112 sub trep {
113   my ($s) = @_ ;
114   my $open = 0;
115   my $p = 0;
116   for (; $p < length($s); $p++) {

```

```

117 $c = substr($s, $p, 1);
118 if ($c eq '}') {
119     if ($open eq 0) {
120         last;
121     }
122     $open--;
123 }
124 if ($c eq '{') {
125     $open++;
126 }
127 }
128 push(@texts, substr($s, 0, $p));
129 return '{TEXT' . (0+@texts - 1) . '}' . substr($s, $p + 1);
130 }
131 $tex =~ s/\\text\{(.+)/trep("$1")/ge;
132 $tex =~ s/(?<=[\s,>()]{0-9A-F}{2}(?:-[0-9A-F]{2})+(?!\\})/|1|/g;
133 $tex =~ s/(?<=[\s,>()]{0-9}+(?:\. [0-9]+)?) (?!\\})/|1|/g;
134 $tex =~ s/([^\{a-zA-Z0-9]|^)\Q(?! [a-zA-Z0-9])\1\\dot{\Phi}/g;
135 $tex =~ s/([^\{a-zA-Z0-9]|^)\Q(?! [a-zA-Z0-9])\1\\Phi/g;
136 $tex =~ s/([^\{a-zA-Z0-9]|^)\Q(?! [a-zA-Z0-9])\1\\bot/g;
137 $tex =~ s/([^\{a-zA-Z0-9]|^)\QD>\1\\Delta{. .>/g;
138 $tex =~ s/([^\{a-zA-Z0-9]|^)\QL>\1\\lambda{. .>/g;
139 $tex =~ s/"([\^"]+)"/|"1"|/g;
140 $tex =~ s/~([0-9]+)/\\alpha_{\1}/g;
141 $tex =~ s/(?::^|(?<=[\s](\\[, .>/)))([a-zA-Z][a-zA-Z0-9]+)(?=[\s](\\[, .-]|$)/|1|/g;
142 $tex =~ s/([^\^]|^)([0-9]+|\\*)\\(\\{[a-z]+|\\{[a-z]+\\}|)
143 (->|\\.\\.\\.>|^>|:=)/\\1\\alpha_{\2}\\vert{\\3\\space{\\4}/xg;
144 $tex =~ s/([^\^]|^)([0-9]+|\\*)
145 (->|\\.\\.\\.>|^>|:=)/\\1\\alpha_{\2}\\space{\\3}/xg;
146 if ($macro ne 'phiq') {
147     if (not $splitting) {
148         $tex =~ s/\\begin{split}\\n\\begin{split}&/g;
149         $tex =~ s/\\n\\s*\\end{split}\\n\\end{split}/g;
150         $tex =~ s/\\n\\n\\n\\n/g;
151         $tex =~ s/\\n\\n\\phiEOL{\\n}/g;
152         $tex =~ s/\\n\\n\\n\\n/g;
153         $tex =~ s/\\n\\n\\n\\n\\n/g;
154         $tex =~ s/([^\&\\s])\s{2}([^\&\\s])/1 \2/g;
155         $tex =~ s/\s{2}/ \\quad/g;
156         $tex = '&' . $tex;
157     }
158     my $lead = '[^\s]+\s(?:->|^>|^:=|^==)\s';
159     my @leads = $tex =~ /&{$lead}/g;
160     my @eols = $tex =~ /&/g;
161     if (0+@leads == 0+@eols && 0+@eols > 1) {
162         $tex =~ s/&{$lead}/1&~/g;
163         $gathered = 0;
164         print "% The \"gathered\" is NOT used because all ' .
165             (0+@eols) . ' lines are ' . (0+@leads) . \" leads\n";
166     }
167 }
168 if ($macro ne 'phiq') {
169     sub strip_tabs {
170         my ($env, $tex) = @_;

```

```

171 $tex =~ s/&//g;
172 return "\\begin{$env}" . $tex . "\\end{$env}";
173 }
174 foreach my $e (('matrix', 'cases')) {
175     $tex =~ s/\\begin{(\Q$e\E\*?)\}(.)\\end{\Q$e\E\*?}/strip_tabs($1, $2)/sge;
176 }
177 }
178 $tex =~ s/\$/\xi{/g;
179 $tex =~ s/(?!{)\^(?!{)/\rho{/g;
180 $tex =~ s/[[\[\\llbracket\\mathbin{/g;
181 $tex =~ s/\\]\]/\mathbin{\\rrbracket{/g;
182 $tex =~ s/TRUE/|TRUE|/g;
183 $tex =~ s/FALSE/|FALSE|/g;
184 $tex =~ s/\?/\varnothing{/g;
185 $tex =~ s/@/\varphi{/g;
186 $tex =~ s/-([a-z]+)/\mathrel{\phiSlot{1}}/g;
187 $tex =~ s/->/\mathbin{\mapsto}/g;
188 $tex =~ s/~>/\mathbin{\phiWave}/g;
189 $tex =~ s/:=/\mathrel{\vDash}/g;
190 $tex =~ s/==/\mathrel{\equiv}/g;
191 $tex =~ s/\.\.\./\mathbin{\phiDotted}/g;
192 $tex =~ s/<</\langle/g;
193 $tex =~ s/>>/\rangle/g;
194 $tex =~ s/|{2,}/|/g;
195 $tex =~ s/|([~|]+)|/\textnormal{\texttt{1}}{/g;
196 $tex =~ s/{TEXT(d+)\}/'\text{' . @texts[$1] . '}/ge;
197 if ($macro eq 'phiq') {
198     print '\(' if ($tex ne '');
199 } else {
200     print '\begin{' , $macro, "}\n";
201     if (not($align)) {
202         if ($gathered) {
203             print '\begin{gathered}' . "\n";
204         } elsif (not $splitting) {
205             print '\begin{split}' . "\n";
206         }
207     }
208 }
209 if ($gathered and not($align)) {
210     $tex =~ s/^&//g;
211     $tex =~ s/\n&/\n/g;
212 }
213 print $tex;
214 if ($macro eq 'phiq') {
215     print '\)' if ($tex ne '');
216 } else {
217     if (not($align)) {
218         if ($gathered) {
219             print "\n" . '\end{gathered}';
220         } elsif (not $splitting) {
221             print "\n" . '\end{split}';
222         }
223     }
224     print "\n" . '\end{' . $macro . '}' ;

```

```

225 }
226 print '\endinput';
227 \end{VerbatimOut}
228 \message{eolang: File with Perl script
229   '\eolang@tmpdir/\jobname-phi.pl' saved^^J}
230 \else
231   \message{eolang: Perl script already exists at
232     "\eolang@tmpdir/\jobname-phi.pl"^^J}
233 \fi
234 \closein 15
235 \fi
236 \makeatother

```

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

```

237 \makeatletter
238 \newcommand\phiSaveTo[1]{\def\eolang@phiSaveTo{#1}}
239 \makeatother

```

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

```

240 \makeatletter
241 \newcommand\eolang@tmp[1]{#1\ifxetex-xe\else\ifluatex-lua\fi\fi.tex}
242 \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:

```

243 \makeatletter
244 \newcommand\eolang@ifabsent[2]{%
245   \IfFileExists
246     {#1}
247     {%
248       \message{eolang: File "#1" already exists ^^J}%
249       \input{#1}}
250   {%
251     \ifdefined\eolang@noshell%
252       \message{eolang: Shell processing is disabled^^J}%
253     \else%
254       \ifnum\ShellEscapeStatus=1\else%
255         \message{eolang: The -shell-escape command line
256           option is not provided, most probably compilation
257           will fail now:^^J}%
258       \fi%
259       #2%
260     \fi%
261   }%
262 }
263 \makeatother

```

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

```

264 \makeatletter\newcommand\eolang@process[1]{
265   \def\hash{\eolang@mdfive
266     {\eolang@tmpdir/\jobname/\eolang@tmp{phi}}-\the\inputlineno}%

```

```

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

```

\phiq Then, we define \phiq command:

```

298 \RequirePackage{xstring}
299 \makeatletter\newcommand\phiq[1]{%
300   \StrSubstitute{\detokenize{#1}}{' '}{'\ '}{\clean}%
301   \def\hash{\pdf@mdfivesum{\clean}-\the\inputlineno}%
302   \ifdefined\eolang@nodollar\else\catcode'\$=3 \fi%
303   \eolang@ifabsent
304     {\eolang@tmpdir/\jobname/\eolang@tmp{\hash-phiq-post}}
305     {%
306       \iexec[log,trace,quiet,stdout=\eolang@tmpdir/\jobname/\eolang@tmp{phiq}]{
307         printf '\%s' '\clean'}%
308       \iexec[quiet,null]{cp "\eolang@tmpdir/\jobname/\eolang@tmp{phiq}"
309         "\eolang@tmpdir/\jobname/\eolang@tmp{\hash-phiq}}}%
310       \iexec[trace,stdout=\eolang@tmpdir/\jobname/\eolang@tmp{\hash-phiq-post}]{
311         perl \eolang@tmpdir/\jobname-phi.pl 'phiq'
312         "\eolang@tmpdir/\jobname/\eolang@tmp{\hash-phiq}"
313         \ifdefined\eolang@nocomments | perl -pe 's/\%.*(\n|$)//g' \fi}%
314       \message{eolang: Parsed 'phiq' at line no. \the\inputlineno^^J}%
315     }%
316   \ifdefined\eolang@nodollar\else\catcode'\$=active\fi%
317 }\makeatother

```


nodollar Then, we redefine dollar sign:

```
318 \ifdefined\eolang@nodollar\else
319   \begingroup
320   \catcode'\$=\active
321   \protected\gdef$#1${\phiq{#1}}
322   \endgroup
323   \AtBeginDocument{\catcode'\$=\active}
324 \fi
```

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

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

```

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

```

```

422 if (exists $opts{'bend'}) {
423   $draw = $draw . 'edge [<F:draw=none><MF:,bend right=' .
424     num($opts{'bend'}) . '>';
425   if (exists $opts{'rho'}) {
426     $draw = $draw . '<MB:,phi-rho>';
427   }
428   $draw = $draw . ']';
429 } else {
430   $draw = $draw . '--';
431 }
432 if (exists $opts{'a'}) {
433   my $a = $opts{'a'};
434   if (index($a, '$') == -1) {
435     $a = '$' . fmt($a) . '$';
436   } else {
437     $a = fmt($a);
438   }
439   $draw = $draw . '<MB: node [phi-attr] {' . $a . '}>';
440 }
441 if (exists $opts{'break'}) {
442   $draw = $draw . '<F: coordinate [pos=' .
443     ($opts{'break'} / 100) . '] (break)>';
444 }
445 $draw = $draw . " (<MF:${to}><B:break-v>";
446 if (exists $opts{'break'}) {
447   print tailor($draw, 'F') . ";\n";
448   print ' \node[outer sep=' . toem(0.1) . 'em,inner sep=0em] ' .
449     'at (break) (break-v) {$' . vertex($to) .
450     '$};' . "\n";
451   print ' ' . tailor($draw, 'B');
452 } else {
453   print tailor($draw, 'M');
454 }
455 } elsif (index($head, '>') >= 0) {
456   my ($from, $to) = split (/=>/, $head);
457   my $size = () = $head =~ /=/g;
458   if ($from eq '') {
459     print '\node [phi-arrow, left=' . toem($size * 0.6) . 'em of ' .
460       $to . '.center]';
461   } elsif ($to eq '') {
462     print '\node [phi-arrow, right=' . toem($size * 0.6) . 'em of ' .
463       $from . '.center]';
464   } else {
465     print '\node [phi-arrow] at ($(' .
466       $from . ')!0.5!(' . $to . ')$)';
467   }
468   print '{}';
469 } elsif (index($head, '!') >= 0) {
470   my ($v, $marker) = split (/!+/, $head);
471   my $size = () = $head =~ /!/g;
472   print '\node [phi-marker, left=' .
473     toem($size * 0.6) . 'em of ' .
474     $v . '.center]{' . fmt($marker) . '}';
475 } elsif (index($head, '+') >= 0) {

```

```

476 my ($v, $suffix) = split (/\/, $head);
477 my @friends = ($v);
478 foreach my $c (@cmds) {
479     $e = $c;
480     $e =~ s/^\s+//g;
481     my $h = $e;
482     $h = substr($e, 0, index($e, ' ')) if index($e, ' ') >= 0;
483     foreach my $f (@friends) {
484         my $add = '';
485         if (index($h, $f . '->') >= 0) {
486             $add = substr($h, index($h, '->') + 2);
487         }
488         if ($h =~ /->\Q${f}\E$/) {
489             $add = substr($h, 0, index($h, '->'));
490         }
491         if (index($e, ' xy:' . $f . ',') >= 0) {
492             $add = $h;
493         }
494         if (index($add, '+') == -1
495             and $add ne ''
496             and not(grep(/^Q${add}\E$/, @friends))) {
497             push(@friends, $add);
498         }
499     }
500 }
501 my @extra = ();
502 foreach my $e (@cmds) {
503     $m = $e;
504     if ($m =~ /\s*\Q${v}\E\s/) {
505         next;
506     }
507     if ($m =~ /\s*[^\s]+\s/ and not($m =~ /\s*\Q${head}\E\s/)) {
508         next;
509     }
510     foreach my $f (@friends) {
511         my $h = $f;
512         $h =~ s/[a-z]$/g;
513         if ($m =~ s/^(s*)\Q${f}\E+\Q${suffix}\E\s?/\1${h}${suffix} /g) {
514             last;
515         }
516         $m =~ s/^(s*)\Q${f}\E\s/\1${h}${suffix} /g;
517         $m =~ s/^(s*)\Q${f}\E->/\1${h}${suffix}->/g;
518         $m =~ s/\sxy:\Q${f}\E,/ xy:${h}${suffix},/g;
519         $m =~ s/->\Q${f}\E\s->${h}${suffix} /g;
520     }
521     if ($m ne $e) {
522         push(@extra, ' ' . $m);
523     }
524 }
525 splice(@extra, 0, 0, @extra[-1]);
526 splice(@extra, -1, 1);
527 splice(@extra, 0, 0, '% clone of ' . $v . ' (' . $head .
528     ')', friends: [' . join(', ', @friends) . '] in ' .
529     (0+@cmds) . ' lines');

```

```

530 splice(@cmds, $c, 1, @extra);
531 print '% cloned ' . $v . ' at line no.' . $c .
532 ' (+ ' . (0+@extra) . ' lines -> ' .
533 (0+@cmds) . ' lines total)';
534 } elsif ($head =~ /^v[0-9]+[a-z]?$/) {
535 print '\node[';
536 if (exists $opts{'xy'}) {
537 my ($v, $right, $down) = split(/,/ , $opts{'xy'});
538 my $loc = '';
539 if ($down > 0) {
540 $loc = 'below ';
541 } elsif ($down < 0) {
542 $loc = 'above ';
543 }
544 if ($right > 0) {
545 $loc = $loc . 'right';
546 } elsif ($right < 0) {
547 $loc = $loc . 'left';
548 }
549 print ', ' . $loc . '=';
550 print toem(abs(num($down))) . 'em and ' .
551 toem(abs(num($right))) . 'em of ' . $v . '.center';
552 }
553 if (exists $opts{'data'}) {
554 print ',phi-data';
555 if ($opts{'data'} ne '') {
556 my $d = $opts{'data'};
557 if (index($d, '|') == -1) {
558 $d = '$\Delta\phiDotted\text{' .
559 '\textnormal{\texttt{' . fmt($d) . '}}}$';
560 } else {
561 $d = fmt($d);
562 }
563 $opts{'box'} = $d;
564 }
565 } elsif (exists $opts{'atom'}) {
566 print ',phi-atom';
567 if ($opts{'atom'} ne '') {
568 my $a = $opts{'atom'};
569 if (index($a, '$') == -1) {
570 $a = '$\lambda\phiDotted{' . fmt($a) . '$';
571 } else {
572 $a = fmt($a);
573 }
574 $opts{'box'} = $a;
575 }
576 } else {
577 print ',phi-object';
578 }
579 if (exists $opts{'edgeless'}) {
580 print ',draw=none';
581 }
582 print ', ' . $opts{'style'} . ']';
583 print ' (' . $head . ')';

```

```

584 print '{';
585 if (exists $opts{'tag'}) {
586   my $t = $opts{'tag'};
587   if (index($t, '$') == -1) {
588     $t = '$' . $t . '$';
589   } else {
590     $t = fmt($t);
591   }
592   print $t;
593 } else {
594   print '$' . vertex($head) . '$';
595 }
596 print '}';
597 if (exists $opts{'box'}) {
598   print ' node[phi-box] at (';
599   print $head, '.south east) {';
600   print $opts{'box'}, '}';
601 }
602 }
603 print ";\n";
604 }
605 print '\end{phicture}%', "\n";
606 print "% --- after processing:\n%";
607 foreach my $c (@cmds) {
608   print '% ', $c, "\n";
609 }
610 print '% --- (' . (0+@cmds) . " lines)\n";
611 print '\endinput';
612 \end{VerbatimOut}
613 \message{eolang: File with Perl script
614   '\eolang@tmpdir/\jobname-sodg.pl' saved^^J}
615 \else
616   \message{eolang: Perl script already exists at
617     "\eolang@tmpdir/\jobname-sodg.pl"^^J}
618 \fi
619 \closein 15
620 \fi
621 \makeatother

```

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

```
622 \setcounter{FancyVerbLine}{0}
```

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

```

623 \RequirePackage{tikz}
624 \usetikzlibrary{arrows}
625 \usetikzlibrary{shapes}
626 \usetikzlibrary{decorations}
627 \usetikzlibrary{decorations.pathmorphing}
628 \usetikzlibrary{decorations.pathreplacing}
629 \usetikzlibrary{positioning}
630 \usetikzlibrary{calc}
631 \usetikzlibrary{math}
632 \usetikzlibrary{arrows.meta}

```

picture Then, we define internal environment phicture:

```

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

```

```

686         \t = atan2(\vy, \vx);
687         if \vx<0 then { \a = 90; } else { \a = -90; };
688         {
689             \draw[arrows={-latex}, decorate,
690                 decoration={%
691                     snake, amplitude=.4mm,
692                     segment length=2mm,
693                     post length=1mm}]
694                 ($(\F)!.5!(\I) +(\t: -\d em) +(\t +\a: 1ex)$)
695                 -- ++(\t: 2*\d em);
696         };
697     }
698 }
699 },
700 decorate
701 }
702 }
703 }
704 \tikzstyle{phi-pi} = [draw,dotted]
705 \tikzstyle{phi-atom} = [phi-object,double]
706 \tikzstyle{phi-box} = [xshift=-5pt,yshift=3pt,draw,fill=white,
707     rectangle,line width=.04em,minimum width=1.2em,anchor=north west,
708     font={\scriptsize}]
709 \tikzstyle{phi-attr} = [midway,sloped,inner sep=0pt,
710     above=2pt,sloped/.append style={transform shape},
711     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:

```

712 \makeatletter
713 \newcommand\sodgSaveTo[1]{\def\eolang@sodgSaveTo{#1}}
714 \makeatother

```

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

```

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

```



```

735   }
736   \catcode'\$\active%
737   \setcounter{FancyVerbLine}{\value{eolang@lineno}}%
738   \def\eolang@sodgSaveTo{\relax}%
739 }\makeatother

```

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

```

740 \RequirePackage{hyperref}
741 \pdfstringdefDisableCommands{
742   \def\({}%
743   \def\)}%
744   \def\alpha{alpha}%
745   \def\varphi{phi}%
746 }
747 \makeatletter
748 \NewExpandableDocumentCommand{\eoAnon}{0{ANONYMIZED}m}{%
749   \ifdefined\eolang@anonymous%
750     \textcolor{orange}{#1}%
751   \else%
752     #2%
753   \fi%
754 }\makeatother

```

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

```

755 \newcommand\eolang{%
756   \eoAnon[XYZ]{\sffamily EO}}

```

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

```

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

```

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

```

759 \newcommand\xmir{%
760   \eoAnon[XML\(^+\)]{XMIR}}

```

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

```

761 \newcommand\phiConst{%
762   \mathrel{\hspace{.15em}}%
763   \mapstochar\mathrel{\hspace{-.15em}}\mapsto}

```

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

```

764 \newcommand\phiWave{%
765   \mapstochar\mathrel{\mspace{0.45mu}}\leadsto}

```

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

```

766 \newcommand\phiSlot[1]{%
767   \xrightarrow{\text{\sffamily\scshape #1}}}

```

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

```
768 \makeatletter
769 \newcommand{\phi0set}[2]{%
770   \mathrel{\mathop{#2}\limits^{
771     \vbox to 0ex{\kern-2\ex@
772       \hbox{\scriptscriptstyle#1}\vss}}}}
773 \newcommand{\phiUset}[2]{%
774   \mathrel{\mathop{#2}\limits_{
775     \vbox to 0ex{\kern-6.3\ex@
776       \hbox{\scriptscriptstyle#1}\vss}}}}
777 \makeatother
```

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

```
778 \newcommand\phiMany[3]{%
779   \phi0set{#3}{\phiUset{#2}{#1}}}
```

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

```
780 \newcommand\phiEOL{\[-4pt]}
```

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

```
781 \RequirePackage{trimclip}
782 \RequirePackage{amsfonts}
783 \makeatletter
784 \newcommand{\phiDotted}{%
785   \mapstochar\mathrel{\mathpalette\phiDotted@\relax}}
786 \newcommand{\phiDotted@}[2]{%
787   \begingroup%
788   \settowidth{\dimen\z@}{\m@th#1\rightarrow}%
789   \settoheight{\dimen\tw@}{\m@th#1\rightarrow}%
790   \sbox\z@{%
791     \makebox[\dimen\z@][s]{%
792       \clipbox{0 0 {0.4\width} 0}%
793       {\resizebox{\dimen\z@}{\height}%
794         {\m@th#1\dashrightarrow}}}%
795     \hss%
796     \clipbox{{0.69\width} {-0.1\height} 0
797       {-\height}}{\m@th#1\rightarrow}%
798   }%
799 }%
800 \ht\z@=\dimen\tw@ \dp\z@=\z@%
801 \box\z@%
802 \endgroup%
803 }
804 \makeatother
```