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

Yegor Bugayenko
`yegor256@gmail.com`

2025/03/26, 0.18.3

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 LaTeX 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

$$
\begin{aligned}
&\texttt{app} \mapsto \llbracket \\
&\quad \rho \phiConst \xi.b.^{2}, \alpha_0|t \phiWave \texttt{TRUE}, \\
&\quad b \mapsto \llbracket \alpha_{*} \mapsto \Phi.\texttt{fn}(56), \\
&\qquad \varphi \mapsto \dot{\Phi}.\texttt{string.trim}(\xi), \\
&\qquad \Delta \phiDotted \texttt{01-FE-C3} \rrbracket \rrbracket, \\
&x \mapsto \llbracket \lambda \phiDotted \varnothing \rrbracket.
\end{aligned}
$$

```
\documentclass{minimal}
\usepackage{eolang}
\begin{document}
\begin{phiquation*}
app -> [[ % it’s abstract!
  ^ !-> $.b.^{^2}, 0/t~> TRUE,
  b -> [[ *-> Q.fn(56),
    @ -> QQ.string.trim($),
    D> 01-FE-C3 ]]]],\\
x -> [[ \lambda ..> ? ]].
\end{phiquation*}
\end{document}
```

`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`),
- “`$`” maps to “ξ” (`\xi`),
- “`&`” maps to “σ” (`\sigma`),
- “`?`” maps to “$\varnothing$” (`\varnothing`),
- “`Q`” maps to “Φ” (`\Phi`),
- “`QQ`” maps to “$\dot{\Phi}$” (`\dot{\Phi}`),
- “`->`” maps to “$\mapsto$” (`\mapsto`),
- “`~>`” maps to “$\phiWave$” (`\phiWave`),
- “`!->`” maps to “$\phiConst$” (`\phiConst`),
- “`..>`” maps to “$\phiDotted$” (`\phiDotted`),
- “`D>`” maps to “$\Delta \phiDotted$” (`\Delta ..>`),
- “`L>`” maps to “$\lambda \phiDotted$” (`\lambda ..>`),
- “`[[`” maps to “$\llbracket$” (`\llbracket`),
- “`]]`” maps to “$\rrbracket$” (`\rrbracket`),
- “`==`” maps to “$\equiv$” (`\equiv`),
- “`|abc|`” maps to “`abc`” (`\texttt{abc}`).

Also, a few symbols are supported for φPU architecture:

- “`<<`” maps to “$\langle$” (`\langle`),
- “`>>`” maps to “$\rangle$” (`\rangle`),
- “`-abc>`” maps to “$\xrightarrow{\text{ABC}}$” (`\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 `\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 “$\alpha_0|\texttt{f} \mapsto x$”. It's also possible to use an asterisk instead of a number, such that `\phiq{*/g->x}` renders as “$\alpha_*|g \mapsto x$”

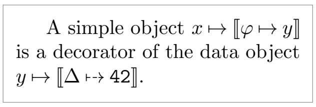
Numbers are automatically converted to fixed-width font, no need to always decorate them with vertical bars.

`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.

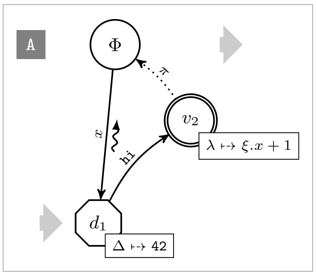
`\phiq` The command `\phiq` lets you inline a φ-calculus expressions using the same simple plain-text notation. You can use dollar sign directly too:



A simple object $x \mapsto [\![\varphi \mapsto y]\!]$ is a decorator of the data object $y \mapsto [\![\Delta \dashrightarrow \texttt{42}]\!]$.

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

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


A
Φ
π
v_2
$\lambda \dashrightarrow \xi.x+1$
x
hi
d_1
$\Delta \dashrightarrow 42$

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

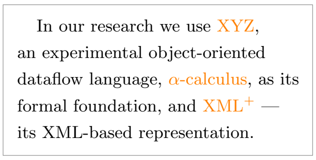
You can also put a marker at the left side of a vertex, using “`v5!A`” syntax, where “`v5`” is the vertex and “`A`” is the text in the marker. They are useful when you put a few graphs on a picture explaining how one graph is transformed to another one and so forth. You can make the distance between 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`
`\phic`
`\xmir`

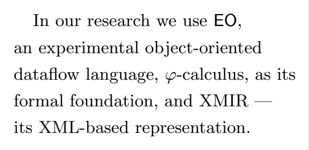
There is also a no-argument command `\eolang` to help you print the name of EO language. It understands the `anonymous` package option and prints itself differently, to double-blind your paper. There is also `\phic` command to print the name of φ-calculus, also sensitive to `anonymous` mode. The macro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



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

```
\usepackage[anonymous]{eolang}
\begin{document}
In our research we use \eolang{}, \\
an experimental object-oriented \\
dataflow language, \phic{}, as its \\
formal foundation, and \xmir{} --- \\
its XML-based representation.
\end{document}
```

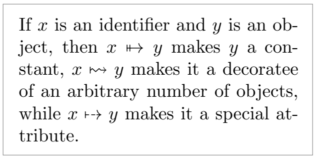
Without the `anonymous` option there will be no orange color:



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

```
\usepackage{eolang}
\begin{document}
In our research we use \eolang{}, \\
an experimental object-oriented \\
dataflow language, \phic{}, as its \\
formal foundation, and \xmir{} --- \\
its XML-based representation.
\end{document}
```

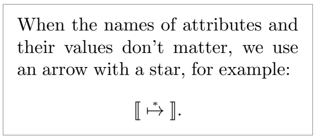
`\phiConst` `\phiWave` `\phiDotted` A few simple commands are defined to help you render arrows. It is recommended not to use them directly, but use `!->` instead. However, if you want to use `\phiConst`, wrap it in `\mathrel` for better display:



If x is an identifier and y is an object, then $x \mathrel{|\!\!\mapsto} y$ makes y a constant, $x \mathrel{\vdash\!\!\rightsquigarrow} y$ makes it a decoratee of an arbitrary number of objects, while $x \mathrel{\vdash\!\!\dashrightarrow} y$ makes it a special attribute.

```
If $x$ is an identifier and $y$ is
an object, then $x \phiConst y$
makes $y$ a constant,
$x \phiWave y$ makes it a decoratee
of an arbitrary number of objects,
while $x \phiDotted y$ makes it
a special attribute.
```

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

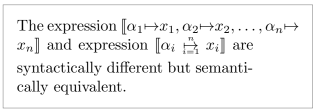


When the names of attributes and their values don't matter, we use an arrow with a star, for example:

$$[\![\overset{*}{\mapsto}]\!].$$

```
When the names of attributes and their
values don't matter, we use an arrow
with a star, for example:
\begin{phiquation*}
[[ \phiOset{*}{->} ]].
\end{phiquation*}
```

`\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 $[\![\alpha_i \overset{n}{\underset{i=1}{\mapsto}} x_i]\!]$ are syntactically different but semantically equivalent.

```
The expression
\phiq{[[ 1-> x_1,
  2-> x_2, \dots,
  \alpha_n -> x_n ]]}
and expression
\phiq{[[ \alpha_i
  \phiMany{->}{i=1}{n} x_i ]]}
are syntactically different but
semantically equivalent.
```

`\phiSaveTo` `\sodgSaveTo` If you want to use `phiquation` or `sodg` environments inside `tabular` or any other environment or command, you won't be able to do this, because `phiquation` and `sodg` are "verbatim" environments. `\phiSaveTo` and `\sodgSaveTo` commands will help you in this situation. You use them right before `\begin{phiquation}` 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:	$[\![x \mapsto \varnothing]\!]$
Bound:	$[\![x \mapsto [\![\Delta \mapstochar\dashrightarrow 42]\!]]\!]$

```
\phiSaveTo{a}
\begin{phiquation*}
[[ x -> [[D>42]] ]]
\end{phiquation*}
\begin{tabular}{p{.5in}l}
Free: & $[[x -> ?]]$ \\
Bound: & \parbox{1in}{\input{a}} \\
\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 substituion 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:

$$\dfrac{x \mapsto [\![\varphi \mapsto y]\!] \quad y \mapsto [\![z \mapsto 42]\!]}{x.z \mapsto 42}\textsf{R1}$$

```
\begin{phiquation*}
\dfrac \
 {x->[[@->y]] \quad y->[[z->42]]} \
 {x.z -> 42} \
 \text{\sffamily R1}
\end{phiquation*}
```

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

$$\dfrac{\begin{split} & x \mapsto \llbracket \varphi \mapsto y, z \mapsto \mathtt{42}, \\ & \quad \alpha_0|g \mapsto \varnothing, \alpha_1|\mathtt{foo} \mapsto \mathtt{42}\rrbracket \end{split}}{\begin{split} & x \mapsto \llbracket \varphi \mapsto y, z \mapsto \varnothing, f \mathrel{\mapstochar\rightsquigarrow} \mathtt{pi}(\\ & \quad \alpha_0 \mapsto \llbracket \psi \mathrel{\mapstochar\mapsto} \mathtt{hello(12)} \rrbracket, \\ & \qquad \alpha_1 \mapsto \mathtt{42})\rrbracket \end{split}}\text{R2}.$$

```
\begin{phiquation*}
\dfrac{\begin{split}
x->[[@->y, z->42,
  0/g->?, 1/foo->42]]
\end{split}}{\begin{split}
x->[[@->y, z->?, f ~> |pi|(
  0->[[ \psi !-> |hello|(12) ]],
    1->42)]]
\end{split}}\text{R2}.
\end{phiquation*}
```

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

$$\mathtt{foo} \mapsto \left\{\begin{matrix} \varnothing \\ \llbracket \lambda \mathrel{\mapstochar\dashrightarrow} \rho \times \xi.\alpha_0 \rrbracket \\ \llbracket \Delta \mathrel{\mapstochar\dashrightarrow} \mathtt{42} \rrbracket \end{matrix}\right\}$$

```
\begin{phiquation*}
foo -> \left\{\begin{matrix} \
? \\
[[ L> ^ \times $.\alpha_0 ]] \\
[[ D> 42 ]] \
\end{matrix}\right\}
\end{phiquation*}
```

The `cases` environment works too:

$$\beta \vDash \begin{cases} [v_2, \varphi \xrightarrow{\text{DTZD}} \mathtt{42}] \\ [v_{33}] \end{cases}$$

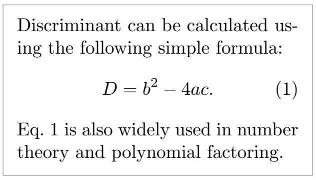
```
\begin{phiquation*}
\beta := \begin{cases} \
[ v_2, @ -dtzd> 42 ] \\
[ v_{33} ] \
\end{cases}
\end{phiquation*}
\end{document}
```

The `phiquation` environment may be used together with the acmart package:

$$\begin{aligned} & x \mapsto \llbracket \\ & \quad y \mapsto \llbracket \\ & \qquad z \mathrel{\mapstochar\mapsto} \xi, f \mathrel{\mapstochar\dashrightarrow} \varnothing \rrbracket\rrbracket, \\ & \beta_1 \vDash [\psi \xrightarrow{\text{WAIT}} \varnothing]. \end{aligned}$$

```
\documentclass{acmart}
\usepackage{eolang}
\thispagestyle{empty}
\begin{document}
\begin{phiquation*}
x -> [[
  y -> [[
    z !-> $, f ..> ? ]]]],\\
\beta_1 := [ \psi -wait> ? ].
\end{phiquation*}
\end{document}
```

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



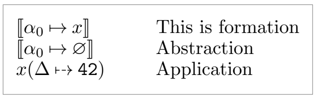
Discriminant can be calculated using the following simple formula:

$$D = b^2 - 4ac. \qquad (1)$$

Eq. 1 is also widely used in number theory and polynomial factoring.

```
Discriminant can be calculated using
the following simple formula:
\begin{phiquation}
D = b{^2} - {4}ac.
\label{d}
\end{phiquation}
Eq.~\ref{d} is also widely used in
number theory and polynomial factoring.
```

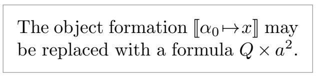
You can add comments to your equations, using the `&&` command (pay attention, the text inside `\text{}` is not processed and treated like a plain text):



$$\begin{aligned}
&[\![\alpha_0 \mapsto x]\!] && \text{This is formation}\\
&[\![\alpha_0 \mapsto \varnothing]\!] && \text{Abstraction}\\
&x(\Delta \mapstochar\dashrightarrow \mathtt{42}) && \text{Application}
\end{aligned}$$

```
\begin{phiquation*}
[[ 0->x ]] && \text{This is formation}
[[ 0->? ]] && \text{Abstraction}
x(D>42) && \text{Application}
\end{phiquation*}
```

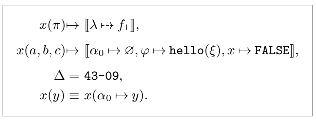
If you don’t use `nodollar` package option, you can still use normal parsing of the dollar sign, by means of `\(...\)` syntax:



The object formation $[\![\alpha_0 \mapsto x]\!]$ may be replaced with a formula $Q \times a^2$.

```
The object formation $[[0->x]]$
may be replaced with a formula
\( Q \times a^2 \).
```

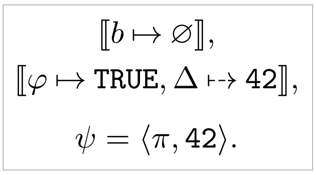
The `phiquation` environment will automatically align formulas by the first arrow, if there are only left-aligned formulas:



$$\begin{aligned}
x(\pi) &\mapsto [\![\lambda \mapstochar\dashrightarrow f_1]\!],\\
x(a,b,c) &\mapsto [\![\alpha_0 \mapsto \varnothing, \varphi \mapsto \mathtt{hello}(\xi), x \mapsto \mathtt{FALSE}]\!],\\
\Delta &= \mathtt{43\text{-}09},\\
x(y) &\equiv x(\alpha_0 \mapsto y).
\end{aligned}$$

```
\begin{phiquation*}
x(\pi) -> [[\lambda ..> f_1]], \\
x(a,b,c) -> [[ \alpha_0 -> ?, \
  @ -> |hello|($), x -> |FALSE| ]], \\
\Delta = |43-09|,
x(y) == x(0-> y).
\end{phiquation*}
```

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



$$[\![b \mapsto \varnothing]\!],$$

$$[\![\varphi \mapsto \mathtt{TRUE}, \Delta \mapstochar\dashrightarrow \mathtt{42}]\!],$$

$$\psi = \langle \pi, \mathtt{42} \rangle.$$

```
\begin{phiquation*}
[[ b -> ? ]],
[[ @ -> TRUE, \Delta ..> 42 ]], \\
\psi = << \pi, 42 >>.
\end{phiquation*}
```

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

$$\begin{split} x(\pi) &\mapsto 4 \\ x(a,b,c) &\mapsto [\![\alpha_0 \mapsto \varnothing]\!] \end{split}$$

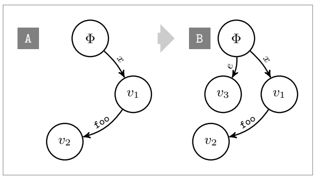
```
\begin{phiquation*}
\begin{split}
x(\pi) & -> 4 \\
x(a,b,c) & -> [[ \alpha_0 -> ? ]] \\
\end{split}
\end{phiquation*}
```

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

$x \mapsto$ `sprintf("Hello, %s!",name)`

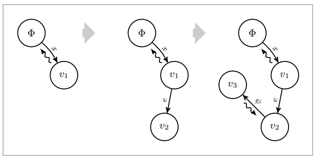
```
\begin{phiquation*}
x -> sprintf("Hello, \%s!", name)
\end{phiquation*}
\end{document}
```

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



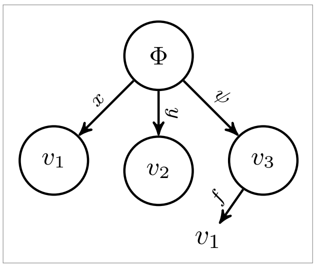
```
\begin{sodg}
v0 \\ v0!!A
v1 xy:v0,.7,1
v0->v1 a:x bend:-10
v2 xy:v1,-1.3,.8
v1->v2 a:|foo| bend:-20
v0+a xy:v0,3,0
v3a xy:v0a,-.7,1
v0a->v3a a:e bend:-15
v0=>v0a \\ v0a!B
\end{sodg}
```

You can make a copy from a copy:



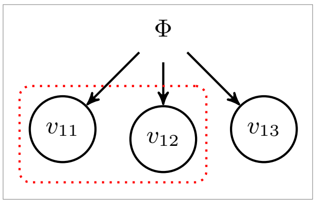
```
\begin{sodg}
v0
v1 xy:v0,.7,1
v0->v1 a:x bend:-10 rho
v0+a xy:v0,3,0 \\ v0=>v0a
v2a xy:v1a,-.8,1.3
v1a->v2a a:e
v0a+b xy:v0a,3,0 \\ v0a=>v0b
v3b xy:v2b,-1,-1
v2b->v3b a:\psi{} rho
\end{sodg}
```

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:



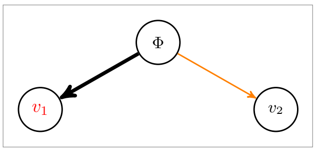
```
\begin{sodg}
v0
v1 xy:v0,-1,1
v0->v1 a:x
v2 xy:v0,0,1
v0->v2 a:y
v3 xy:v0,1,1
v0->v3 a:\psi{}
v3->v1 a:f bend:-75 break:30
\end{sodg}
```

You can add TikZ commands to `sodg` graph, for example:



```
\begin{sodg}
v0 edgeless
v11 xy:v0,-1,1 \\ v0->v11
v12 xy:v0,0,1 \\ v0->v12
v13 xy:v0,1,1 \\ v0->v13
\node[draw=red,rounded corners,\
 dotted,fit=(v11) (v12)] {};
\end{sodg}
```

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



```
\begin{sodg}
v0
v1 xy:v0,-2,1 style:font=\color{red}
v2 xy:v0,2,1
v0->v1 style:line width=2pt
v0->v2 style:draw=orange
\end{sodg}
```

4 Implementation

First, we include a few packages. We need stmaryrd for `\llbracket` and `\rrbracket` commands:

```
\RequirePackage{stmaryrd}
```

We need amsmath for `equation*` environment:

```
\RequirePackage{amsmath}
```

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

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

We need fancyvrb for `\VerbatimEnvironment` command:

```
\RequirePackage{fancyvrb}
```

We need iexec for executing Perl scripts:

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

Then, we process package options:

```
\RequirePackage{pgfopts}
\RequirePackage{ifluatex}
\RequirePackage{ifxetex}
\pgfkeys{
  /eolang/.cd,
  tmpdir/.store in=\eolang@tmpdir,
  tmpdir/.default=_eolang\ifxetex-xe\else\ifluatex-lua\fi\fi,
  nocomments/.store in=\eolang@nocomments,
  anonymous/.store in=\eolang@anonymous,
  noshell/.store in=\eolang@noshell,
  tmpdir
}
\ProcessPgfPackageOptions{/eolang}
```

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

```
\makeatletter
\ifdefined\eolang@noshell\else\RequirePackage{shellesc}\fi
\IfFileExists
  {\eolang@tmpdir/\jobname}
  {\message{eolang: Temporary directory "\eolang@tmpdir/\jobname"
    already exists^^J}}
  {
    \ifdefined\eolang@noshell
      \message{eolang: Temporary directory "\eolang@tmpdir/\jobname"
        is not created, because of the "noshell" package option,
        most probably the compilation will fail later^^J}
    \else
      \ifnum\ShellEscapeStatus=1
        \iexec[null]{mkdir -p "\eolang@tmpdir/\jobname"}
      \else
        \message{eolang: Temporary directory "\eolang@tmpdir/\jobname"
          is not created, because -shell-escape is not set, and
          it doesn't exist, most probably the compilation
          will fail later^^J}
      \fi
    \fi
  }
\makeatother
```

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

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

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

```
\RequirePackage{pdftexcmds}
\makeatletter
\newcommand\eolang@mdfive[1]{\pdf@filemdfivesum{#1}}
\makeatother
```

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

```
\makeatletter
\ifdefined\eolang@noshell
  \message{eolang: Perl script is not going to be created,
  at "\eolang@tmpdir/\jobname-phi.pl" because of the "noshell"
  package option^^J}
\else
\openin 15=\eolang@tmpdir/\jobname-phi.pl
\ifeof 15
\message{eolang: Perl script is going to be created,
  because it is absent at "\eolang@tmpdir/\jobname-phi.pl",
  but if -shell-escape is not set, the compilation will
  most likely fail now^^J}
\begin{VerbatimOut}{\eolang@tmpdir/\jobname-phi.pl}
$macro = $ARGV[0];
open(my $fh, '<', $ARGV[1]);
my $tex; { local $/; $tex = <$fh>; }
print "% This file is auto-generated by eolang.sty 0.18.3\n";
print '% There are ', length($tex),
  ' chars in the input: ', $ARGV[1], "\n";
print '% ---', "\n";
if (index($tex, "\t") > 0) {
  print "TABS are prohibited!";
  exit 1;
}
my @lines = split (/\n/g, $tex);
foreach my $t (@lines) {
  print '% ', $t, "\n";
}
print '% ---', "\n";
$tex =~ s/(?<!\\)%.*\n/\n/g;
$tex =~ s/^\s+|\s+$//g;
my $splitting = $tex =~ /^\\begin\{split\}/;
if ($splitting) {
  print '% The manual splitting mode is ON since \begin{split} started the text' . "\n";
}
my $indents = $tex =~ /\n +/g;
my $gathered = (0 == $indents);
if ($gathered) {
  if ($splitting) {
    print '% The "gathered" is NOT used because of manual splitting' . "\n";
    $gathered = 0;
  } else {
    print '% The "gathered" is used since all lines are left-aligned' . "\n";
  }
} else {
  print '% The "gathered" is NOT used because ' .
    $indents . " lines are indented\n";
}
my $align = 0;
print '% The "align" is NOT used by default' . "\n";
if (index($tex, '&&') >= 0) {
  $macro =~ s/equation/align/g;
  $align = 1;
  print '% The "align" is used because of && seen in the text' . "\n";
}
if ($macro ne 'phiq') {
  if (not $splitting) {
    $tex =~ s/\\\\\n/\n\n/g;
    $tex =~ s/\\\n\s*//g;
  }
  $tex =~ s/\n*(\\label\{[^\}]+\})\n*/\1/g;
  $tex =~ s/\n{3,}/\n\n/g;
}
my @texts = ();
sub trep {
  my ($s) = @_;
  my $open = 0;
  my $p = 0;
  for (; $p < length($s); $p++) {
    $c = substr($s, $p, 1);
    if ($c eq ’}’) {
      if ($open eq 0) {
        last;
      }
      $open--;
    }
    if ($c eq ’{’) {
      $open++;
    }
  }
  push(@texts, substr($s, 0, $p));
  return ’{TEXT’ . (0+@texts - 1) . ’}’ . substr($s, $p + 1);
}
$tex =~ s/\\text\{(.+)/trep("$1")/ge;
if (not $splitting) {
  $tex =~ s/(?<![{&])&(?![&}])/\\sigma{}/g;
}
$tex =~ s/([^\\{a-z0-9]|^)QQ(?![a-z0-9])/\1\\dot{\\Phi{}}/g;
$tex =~ s/([^\\{a-z0-9]|^)Q(?![a-z0-9])/\1\\Phi{}/g;
$tex =~ s/([^\\{a-z0-9]|^)D>/\1\\Delta{}..>/g;
$tex =~ s/([^\\{a-z0-9]|^)L>/\1\\lambda{}..>/g;
$tex =~ s/"([^"]+)"/|"\1"|/g;
$tex =~ s/(^|(?<=[\s)(\]\[,.>\/]))([a-zA-Z][a-z0-9]+)(?=[\s)(\]\[,.-]|$)/|\2|/g;
$tex =~ s/([^_^]|^)([0-9]+|\*)\/(\\?[a-z]+|\|[a-z]+\|)
  (->|\.\.>|~>|:=|!->)/\1\\alpha_{\2}\\vert{}\3\\space{}\4/xg;
$tex =~ s/([^_^]|^)([0-9]+|\*)
  (->|\.\.>|~>|:=|!->)/\1\\alpha_{\2}\\space{}\3/xg;
if ($macro ne ’phiq’) {
  if (not $splitting) {
    $tex =~ s/\\begin\{split\}\n/\\begin{split}&/g;
    $tex =~ s/\n\s*\\end\{split\}/\\end{split}/g;
    $tex =~ s/\n\n/\\\\&/g;
    $tex =~ s/\n/\\phiEOL{}\n&/g;
    $tex =~ s/\\\\$//g;
    $tex =~ s/\\\\/\\\\\n/g;
    $tex =~ s/([^&\s])\s{2}([^\s])/\1 \2/g;
    $tex =~ s/\s{2}/ \\quad{}/g;
    $tex = ’&’ . $tex;
  }
  my $lead = ’[^\s]+\s(?:->|:=|=|==)\s’;
  my @leads = $tex =~ /&${lead}/g;
  my @eols = $tex =~ /&/g;
  if (0+@leads == 0+@eols && 0+@eols > 1) {
    $tex =~ s/&(${lead})/\1&~/g;
    $gathered = 0;
    print ’% The "gathered" is NOT used because all ’ .
      (0+@eols) . ’ lines are ’ . (0+@leads) . " leads\n";
  }
}
if ($macro ne ’phiq’) {
  sub strip_tabs {
    my ($env, $tex) = @_;
    $tex =~ s/&//g;
    return "\\begin{$env}" . $tex . "\\end{$env}";
  }
  foreach my $e ((’matrix’, ’cases’)) {
    $tex =~ s/\\begin\{(\Q$e\E\*?)\}(.+)\\end\{\Q$e\E\*?\}/strip_tabs($1, $2)/sge;
  }
}
$tex =~ s/\$/\\xi{}/g;
$tex =~ s/(?<!\{)\^(?!\{)/\\rho{}/g;
$tex =~ s/\[\[/\\llbracket\\mathbin{}/g;
$tex =~ s/\]\]/\\mathbin{}\\rrbracket{}/g;
$tex =~ s/([\s,>(])([0-9A-F]{2}(?:-[0-9A-F]{2})+|
  [0-9]+(?:\.[0-9]+)?)(?!\{)/\1|\2|/xg;
$tex =~ s/TRUE/|TRUE|/g;
$tex =~ s/FALSE/|FALSE|/g;
$tex =~ s/\?/\\varnothing{}/g;
$tex =~ s/@/\\varphi{}/g;
$tex =~ s/-([a-z]+)>/\\mathrel{\\phiSlot{\1}}/g;
$tex =~ s/!->/\\mathbin{\\phiConst}/g;
$tex =~ s/->/\\mathbin{\\mapsto}/g;
$tex =~ s/~>/\\mathbin{\\phiWave}/g;
$tex =~ s/:=/\\mathrel{\\vDash}/g;
$tex =~ s/==/\\mathrel{\\equiv}/g;
$tex =~ s/\.\.>/\\mathbin{\\phiDotted}/g;
$tex =~ s/<</\\langle/g;
$tex =~ s/>>/\\rangle/g;
$tex =~ s/\|{2,}/|/g;
$tex =~ s/\|([^\|]+)\|/\\textnormal{\\texttt{\1}}{}/g;
$tex =~ s/\{TEXT(\d+)\}/’\\text{’ . @texts[$1] . ’}’;/ge;
if ($macro eq ’phiq’) {
  print ’\(’ if ($tex ne ’’);
} else {
  print ’\begin{’, $macro, "}\n";
  if (not($align)) {
    if ($gathered) {
      print ’\begin{gathered}’ . "\n";
    } elsif (not $splitting) {
      print ’\begin{split}’ . "\n";
    }
  }
}
if ($gathered and not($align)) {
  $tex =~ s/^&//g;
  $tex =~ s/\n&/\n/g;
}
print $tex;
if ($macro eq ’phiq’) {
  print ’\)’ if ($tex ne ’’);
} else {
  if (not($align)) {
    if ($gathered) {
      print "\n" . ’\end{gathered}’;
    } elsif (not $splitting) {
      print "\n" . ’\end{split}’;
    }
  }
  print "\n" . ’\end{’ . $macro . ’}’;
}
print ’\endinput’;
\end{VerbatimOut}
\message{eolang: File with Perl script
  ’\eolang@tmpdir/\jobname-phi.pl’ saved^^J}
\else
  \message{eolang: Perl script already exists at
    "\eolang@tmpdir/\jobname-phi.pl"^^J}
\fi
\closein 15
\fi
\makeatother
```

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

```
\makeatletter
\newcommand\phiSaveTo[1]{\def\eolang@phiSaveTo{#1}}
\makeatother
```

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

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

```
\makeatletter
\newcommand\eolang@ifabsent[2]{%
  \IfFileExists
    {#1}
    {%
      \message{eolang: File "#1" already exists ^^J}%
      \input{#1}}
    {%
      \ifdefined\eolang@noshell%
        \message{eolang: Shell processing is disabled^^J}%
      \else%
        \ifnum\ShellEscapeStatus=1\else%
          \message{eolang: The -shell-escape command line
          option is not provided, most probably compilation
          will fail now:^^J}%
        \fi%
        #2%
      \fi%
    }%
}
\makeatother
```

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

```
\makeatletter\newcommand\eolang@process[1]{
  \def\hash{\eolang@mdfive
    {\eolang@tmpdir/\jobname/\eolang@tmp{phiquation}}-\the\inputlineno}%
  \eolang@ifabsent
    {\eolang@tmpdir/\jobname/\eolang@tmp{\hash-phiquation-post}}
    {%
      \iexec[null]{cp "\eolang@tmpdir/\jobname/\eolang@tmp{phiquation}"
        "\eolang@tmpdir/\jobname/\eolang@tmp{\hash-phiquation}"}%
      \message{Start parsing 'phi' at line no. \the\inputlineno^^J}
      \iexec[trace,stdout=\eolang@tmpdir/\jobname/\eolang@tmp{\hash-phiquation-post}]{
        perl "\eolang@tmpdir/\jobname-phi.pl"
        '#1'
        "\eolang@tmpdir/\jobname/\eolang@tmp{\hash-phiquation}"
        \ifdefined\eolang@nocomments | perl -pe 's/\%.*(\\n|$)//g'\fi
        \ifdefined\eolang@phiSaveTo > \eolang@phiSaveTo\fi}%
    }%
  \setcounter{FancyVerbLine}{\value{eolang@lineno}}%
  \def\eolang@phiSaveTo{\relax}%
}
%
\newenvironment{phiquation*}%
{\catcode`\|=12 \VerbatimEnvironment%
\setcounter{eolang@lineno}{\value{FancyVerbLine}}%
\begin{VerbatimOut}
  {\eolang@tmpdir/\jobname/\eolang@tmp{phiquation}}}
{\end{VerbatimOut}\eolang@process{equation*}}
%
\newenvironment{phiquation}%
{\catcode`\|=12 \VerbatimEnvironment%
\setcounter{eolang@lineno}{\value{FancyVerbLine}}%
\begin{VerbatimOut}
  {\eolang@tmpdir/\jobname/\eolang@tmp{phiquation}}}
{\end{VerbatimOut}\eolang@process{equation}}
\makeatother
```

\phiq Then, we define \phiq command:

```
\RequirePackage{xstring}
\makeatletter\newcommand\phiq[1]{%
  \StrSubstitute{\detokenize{#1}}{'}{'"'"'}[\clean]%
  \def\hash{\pdf@mdfivesum{\clean}-\the\inputlineno}%
  \ifdefined\eolang@nodollar\else\catcode`\$=3 \fi%
  \eolang@ifabsent
    {\eolang@tmpdir/\jobname/\eolang@tmp{\hash-phiq-post}}
    {%
```

```
      \iexec[log,trace,quiet,stdout=\eolang@tmpdir/\jobname/\eolang@tmp{phiq}]{
        printf ’\%s’ ’\clean’}%
      \iexec[quiet,null]{cp "\eolang@tmpdir/\jobname/\eolang@tmp{phiq}"
        "\eolang@tmpdir/\jobname/\eolang@tmp{\hash-phiq}"}%
      \iexec[trace,stdout=\eolang@tmpdir/\jobname/\eolang@tmp{\hash-phiq-post}]{
        perl \eolang@tmpdir/\jobname-phi.pl ’phiq’
        "\eolang@tmpdir/\jobname/\eolang@tmp{\hash-phiq}"
        \ifdefined\eolang@nocomments | perl -pe ’s/\%.*(\\n|$)//g’ \fi}%
      \message{eolang: Parsed ‘phiq’ at line no. \the\inputlineno^^J}%
    }%
  \ifdefined\eolang@nodollar\else\catcode‘\$\active\fi%
}\makeatother
```

`nodollar` Then, we redefine dollar sign:

```
\ifdefined\eolang@nodollar\else
  \begingroup
  \catcode‘\$=\active
  \protected\gdef$#1${\phiq{#1}}
  \endgroup
  \AtBeginDocument{\catcode‘\$=\active}
\fi
```

`-sodg.pl` Then, we create a Perl script for sodg graphs processing using `VerbatimOut` from fancyvrb:

```
\makeatletter
\ifdefined\eolang@noshell
\message{eolang: Perl script is not going to be created
  at "\eolang@tmpdir/\jobname-sodg.pl", because of the
  "noshell" package option^^J}
\else
\openin 15=\eolang@tmpdir/\jobname-sodg.pl
\ifeof 15
\message{eolang: Perl script is going to be created,
  because it is absent at "\eolang@tmpdir/\jobname-sodg.pl",
  but if -shell-escape is not set, the compilation will
  most likely fail now^^J}
\begin{VerbatimOut}{\eolang@tmpdir/\jobname-sodg.pl}
sub num {
  my ($i) = @_;
  $i =~ s/(\+|-)\./\10./g;
  return $i;
}
sub fmt {
  my ($tex) = @_;
  $tex =~ s/\|([^\|]+)\|/\\textnormal{\\texttt{\1}}/g;
  return $tex;
}
sub toem {
  my ($cm) = @_;
  return $cm * 2.8;
}
sub vertex {
  my ($v) = @_;
  if (index($v, ’v0’) == 0) {
    return '\Phi';
  } else {
    $v =~ s/^v/v_{/g;
    $v =~ s/[^0-9]$//g;
    return $v . '}';
  }
}
sub tailor {
  my ($t, $m) = @_;
  $t =~ s/<([A-Z]?${m}[A-Z]?):([^>]+)>/\2/g;
  $t =~ s/<[A-Z]+:[^>]+>//g;
  return $t;
}
open(my $fh, '<', $ARGV[0]);
my $tex; { local $/; $tex = <$fh>; }
if (index($tex, "\t") > 0) {
  print "TABS are prohibited!";
  exit 1;
}
print '% This file is auto-generated', "\n%\n";
print '% --- there are ', length($tex),
  ' chars in the input (', $ARGV[0], "):\n";
foreach my $t (split (/\n/g, $tex)) {
  print '% ', $t, "\n";
}
print "% ---\n";
$tex =~ s/\\\\/\n/g;
$tex =~ s/\\\n//g;
$tex =~ s/(\\[a-zA-Z]+)\s+/\1/g;
$tex =~ s/\n{2,}/\n/g;
my @cmds = split(/\n/g, $tex);
print '% --- before processing:' . "\n";
foreach my $t (split (/\n/g, $tex)) {
  print '% ', $t, "\n";
}
print '% ---';
print ' (' . (0+@cmds) . " lines)\n";
print '\begin{phicture}', "\n";
for (my $c = 0; $c < 0+@cmds; $c++) {
  my $cmd = $cmds[$c];
  $cmd =~ s/^\s+//g;
  $cmd =~ s/(?<!\\)%.*//g;
  my ($head, $tail) = split(/ /, $cmd, 2);
  my %opts = {};
  my ($body, $style) = split(/style:/, $tail, 2);
  $opts{'style'} = $style;
  $tail = $body;
  foreach my $p (split(/ /, $tail)) {
    my ($q, $t) = split(/:/, $p);
    $opts{$q} = $t;
  }
  if (index($head, '\\') == 0) {
    print $cmd;
  } elsif (index($head, '->') >= 0) {
    my $draw = '\draw[';
    if (exists $opts{'pi'}) {
      $draw = $draw . '<MB:phi-pi><F:draw=none>';
      if (not exists $opts{'a'}) {
        $opts{'a'} = '\pi';
      }
    }
    if (exists $opts{'rho'} and not(exists $opts{'bend'})) {
      $draw = $draw . '<MB:,phi-rho>';
    }
    $draw = $draw . ',' . $opts{'style'} . ']';
    my ($from, $to) = split (/->/, $head);
    $draw = $draw . " (${from}) ";
    if (exists $opts{'bend'}) {
      $draw = $draw . 'edge [<F:draw=none><MF:,bend right=' .
        num($opts{'bend'}) . '>';
      if (exists $opts{'rho'}) {
        $draw = $draw . '<MB:,phi-rho>';
      }
      $draw = $draw . ']';
    } else {
      $draw = $draw . '--';
    }
    if (exists $opts{'a'}) {
      my $a = $opts{'a'};
      if (index($a, '$') == -1) {
        $a = '$' . fmt($a) . '$';
      } else {
        $a = fmt($a);
      }
      $draw = $draw . '<MB: node [phi-attr] {' . $a . '}>';
    }
    if (exists $opts{'break'}) {
      $draw = $draw . '<F: coordinate [pos=' .
        ($opts{'break'} / 100) . '] (break)>';
    }
    $draw = $draw . " (<MF:${to}><B:break-v>)";
    if (exists $opts{'break'}) {
      print tailor($draw, 'F') . ";\n";
      print '  \node[outer sep=' . toem(0.1) . 'em,inner sep=0em] ' .
        'at (break) (break-v) {$' . vertex($to) .
        '$};' . "\n";
      print '  ' . tailor($draw, 'B');
    } else {
      print tailor($draw, 'M');
    }
  } elsif (index($head, '=>') >= 0) {
    my ($from, $to) = split (/=+>/, $head);
    my $size = () = $head =~ /=/g;
    if ($from eq '') {
      print '\node [phi-arrow, left=' . toem($size * 0.6) . 'em of ' .
        $to . '.center]';
    } elsif ($to eq '') {
      print '\node [phi-arrow, right=' . toem($size * 0.6) . 'em of ' .
      $from . ’.center]’;
  } else {
    print ’\node [phi-arrow] at ($(’ .
      $from . ’)!0.5!(’ . $to . ’)$)’;
  }
  print ’{}’;
} elsif (index($head, ’!’) >= 0) {
  my ($v, $marker) = split (/!+/, $head);
  my $size = () = $head =~ /!/g;
  print ’\node [phi-marker, left=’ .
    toem($size * 0.6) . ’em of ’ .
    $v . ’.center]{’ . fmt($marker) . ’}’;
} elsif (index($head, ’+’) >= 0) {
  my ($v, $suffix) = split (/\+/, $head);
  my @friends = ($v);
  foreach my $c (@cmds) {
    $e = $c;
    $e =~ s/^\s+//g;
    my $h = $e;
    $h = substr($e, 0, index($e, ’ ’)) if index($e, ’ ’) >= 0;
    foreach my $f (@friends) {
      my $add = ’’;
      if (index($h, $f . ’->’) >= 0) {
        $add = substr($h, index($h, ’->’) + 2);
      }
      if ($h =~ /->\Q${f}\E$/) {
        $add = substr($h, 0, index($h, ’->’));
      }
      if (index($e, ’ xy:’ . $f . ’,’) >= 0) {
        $add = $h;
      }
      if (index($add, ’+’) == -1
        and $add ne ’’
        and not(grep(/^\Q${add}\E$/, @friends))) {
        push(@friends, $add);
      }
    }
  }
  my @extra = ();
  foreach my $e (@cmds) {
    $m = $e;
    if ($m =~ /^\s*\Q${v}\E\s/) {
      next;
    }
    if ($m =~ /^\s*[^\s]+\+/ and not($m =~ /^\s*\Q${head}\E\s/)) {
      next;
    }
    foreach my $f (@friends) {
      my $h = $f;
      $h =~ s/[a-z]$//g;
      if ($m =~ s/^(\s*)\Q${f}\E\+\Q${suffix}\E\s?/\1${h}${suffix} /g) {
        last;
      }
      $m =~ s/^(\s*)\Q${f}\E\s/\1${h}${suffix} /g;
      $m =˜ s/ˆ(\s*)\Q${f}\E->/\1${h}${suffix}->/g;
      $m =˜ s/\sxy:\Q${f}\E,/ xy:${h}${suffix},/g;
      $m =˜ s/->\Q${f}\E\s/->${h}${suffix} /g;
    }
    if ($m ne $e) {
      push(@extra, ’ ’ . $m);
    }
  }
  splice(@extra, 0, 0, @extra[-1]);
  splice(@extra, -1, 1);
  splice(@extra, 0, 0, ’% clone of ’ . $v . ’ (’ . $head .
    ’), friends: [’ . join(’, ’, @friends) . ’] in ’ .
    (0+@cmds) . ’ lines’);
  splice(@cmds, $c, 1, @extra);
  print ’% cloned ’ . $v . ’ at line no.’ . $c .
    ’ (+’ . (0+@extra) . ’ lines -> ’ .
    (0+@cmds) . ’ lines total)’;
} elsif ($head =˜ /ˆv[0-9]+[a-z]?$/) {
  print ’\node[’;
  if (exists $opts{’xy’}) {
    my ($v, $right, $down) = split(/,/, $opts{’xy’});
    my $loc = ’’;
    if ($down > 0) {
      $loc = ’below ’;
    } elsif ($down < 0) {
      $loc = ’above ’;
    }
    if ($right > 0) {
      $loc = $loc . ’right’;
    } elsif ($right < 0) {
      $loc = $loc . ’left’;
    }
    print ’,’ . $loc . ’=’;
    print toem(abs(num($down))) . ’em and ’ .
      toem(abs(num($right))) . ’em of ’ . $v . ’.center’;
  }
  if (exists $opts{’data’}) {
    print ’,phi-data’;
    if ($opts{’data’} ne ’’) {
      my $d = $opts{’data’};
      if (index($d, ’|’) == -1) {
        $d = ’$\Delta\phiDotted\text{’ .
          ’\textnormal{\texttt{’ . fmt($d) . ’}}}$’;
      } else {
        $d = fmt($d);
      }
      $opts{’box’} = $d;
    }
  } elsif (exists $opts{’atom’}) {
    print ’,phi-atom’;
    if ($opts{’atom’} ne ’’) {
      my $a = $opts{’atom’};
      if (index($a, ’$’) == -1) {
        $a = ’$\lambda\phiDotted{}’ . fmt($a) . ’$’;
        } else {
          $a = fmt($a);
        }
        $opts{'box'} = $a;
      }
    } else {
      print ',phi-object';
    }
    if (exists $opts{'edgeless'}) {
      print ',draw=none';
    }
    print ',' . $opts{'style'} . ']';
    print ' (' . $head . ')';
    print ' {';
    if (exists $opts{'tag'}) {
      my $t = $opts{'tag'};
      if (index($t, '$') == -1) {
        $t = '$' . $t . '$';
      } else {
        $t = fmt($t);
      }
      print $t;
    } else {
      print '$' . vertex($head) . '$';
    }
    print '}';
    if (exists $opts{'box'}) {
      print ' node[phi-box] at (';
      print $head, '.south east) {';
      print $opts{'box'}, '}';
    }
  }
  print ";\n";
}
print '\end{phicture}%', "\n";
print "% --- after processing:\n%";
foreach my $c (@cmds) {
  print '% ', $c, "\n";
}
print '% --- (' . (0+@cmds) . " lines)\n";
print '\endinput';
\end{VerbatimOut}
\message{eolang: File with Perl script
  '\eolang@tmpdir/\jobname-sodg.pl' saved^^J}
\else
  \message{eolang: Perl script already exists at
    "\eolang@tmpdir/\jobname-sodg.pl"^^J}
\fi
\closein 15
\fi
\makeatother
```

`FancyVerbLine` Then, we reset the counter for fancyvrb, so that it starts counting lines from zero when the document starts rendering:

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

tikz Then, we include tikz package and its libraries:

```
\RequirePackage{tikz}
  \usetikzlibrary{arrows}
  \usetikzlibrary{shapes}
  \usetikzlibrary{decorations}
  \usetikzlibrary{decorations.pathmorphing}
  \usetikzlibrary{decorations.pathreplacing}
  \usetikzlibrary{positioning}
  \usetikzlibrary{calc}
  \usetikzlibrary{math}
  \usetikzlibrary{arrows.meta}
```

phicture Then, we define internal environment `phicture`:

```
\newenvironment{phicture}%
  {\noindent\begin{tikzpicture}[
    ->,>=stealth',node distance=0,line width=.08em,
    pics/parallel arrow/.style={
      code={\draw[-latex,phi-rho] (##1) -- (-##1);}}]}%
  {\end{tikzpicture}}
\tikzstyle{phi-arrow} = [fill=white!80!black, single arrow,
  minimum height=0.05em, minimum width=0.05em,
  single arrow head extend=2mm]
\tikzstyle{phi-marker} = [inner sep=0pt, minimum height=1.4em,
  minimum width=1.4em, font={\small\color{white}\ttfamily},
  fill=gray]
\tikzstyle{phi-thing} = [inner sep=0pt,minimum height=2.4em,
  draw,font={\small}]
\tikzstyle{phi-object} = [phi-thing,circle]
\tikzstyle{phi-data} = [phi-thing,regular polygon,
  regular polygon sides=8]
\tikzstyle{phi-empty} = [phi-object]
\tikzset{%
  phi-rho/.style={
    postaction={%
      decoration={
        show path construction,
        curveto code={
          \tikzmath{
            coordinate \I, \F, \v;
            \I = (\tikzinputsegmentfirst);
            \F = (\tikzinputsegmentlast);
            \v = ($(\I) -(\F)$);
            real \d, \a, \r, \t;
            \d = 0.8;
            \t = atan2(\vy, \vx);
            if \vx<0 then { \a = 90; } else { \a = -90; };
            {
              \draw[arrows={-latex}, decorate,
              decoration={%
                snake, amplitude=.4mm,
                segment length=2mm,
                post length=1mm
              }]
              ($(\F)!.5!(\I) +(\t: -\d em) +(\t +\a: 1ex)$)
              -- ++(\t: 2*\d em);
            };
          }
        },
        lineto code={
          \tikzmath{
            coordinate \I, \F, \v;
            \I = (\tikzinputsegmentfirst);
            \F = (\tikzinputsegmentlast);
            \v = ($(\I) -(\F)$);
            real \d, \a, \r, \t;
            \d = 0.8;
            \t = atan2(\vy, \vx);
            if \vx<0 then { \a = 90; } else { \a = -90; };
            {
              \draw[arrows={-latex}, decorate,
              decoration={%
                snake, amplitude=.4mm,
                segment length=2mm,
                post length=1mm}]
              ($(\F)!.5!(\I) +(\t: -\d em) +(\t +\a: 1ex)$)
              -- ++(\t: 2*\d em);
            };
          }
        }
      },
      decorate
    }
  }
}
\tikzstyle{phi-pi} = [draw,dotted]
\tikzstyle{phi-atom} = [phi-object,double]
\tikzstyle{phi-box} = [xshift=-5pt,yshift=3pt,draw,fill=white,
  rectangle,line width=.04em,minimum width=1.2em,anchor=north west,
  font={\scriptsize}]
\tikzstyle{phi-attr} = [midway,sloped,inner sep=0pt,
  above=2pt,sloped/.append style={transform shape},
  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:

```
\makeatletter
\newcommand\sodgSaveTo[1]{\def\eolang@sodgSaveTo{#1}}
\makeatother
```

sodg Then, we create a new environment sodg, as suggested here:

```
\makeatletter\newenvironment{sodg}%
{\catcode`\|=12 \VerbatimEnvironment%
\setcounter{eolang@lineno}{\value{FancyVerbLine}}%
\begin{VerbatimOut}
  {\eolang@tmpdir/\jobname/\eolang@tmp{sodg}}}
{\end{VerbatimOut}%
```

```
  \def\hash{\eolang@mdfive
    {\eolang@tmpdir/\jobname/\eolang@tmp{sodg}}-\the\inputlineno}%
  \catcode‘\$=3 %
  \eolang@ifabsent
    {\eolang@tmpdir/\jobname/\eolang@tmp{\hash-sodg-post}}
    {%
      \iexec[null]{cp "\eolang@tmpdir/\jobname/\eolang@tmp{sodg}"
        "\eolang@tmpdir/\jobname/\eolang@tmp{\hash-sodg}"}%
      \message{eolang: Start parsing ‘sodg’ at line no. \the\inputlineno^^J}
      \iexec[trace,stdout=\eolang@tmpdir/\jobname/\eolang@tmp{\hash-sodg-post}]{
        perl "\eolang@tmpdir/\jobname-sodg.pl"
        "\eolang@tmpdir/\jobname/\eolang@tmp{\hash-sodg}"
        \ifdefined\eolang@nocomments | perl -pe ’s/\%.*(\\n|$)//g’\fi
        \ifdefined\eolang@sodgSaveTo > \eolang@sodgSaveTo\fi}%
    }
  \catcode‘\$\active%
  \setcounter{FancyVerbLine}{\value{eolang@lineno}}%
  \def\eolang@sodgSaveTo{\relax}%
}\makeatother
```

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

```
\RequirePackage{hyperref}
\pdfstringdefDisableCommands{
  \def\({}%
  \def\){}%
  \def\alpha{alpha}%
  \def\varphi{phi}%
}
\makeatletter
\NewExpandableDocumentCommand{\eoAnon}{O{ANONYMIZED}m}{%
  \ifdefined\eolang@anonymous%
    \textcolor{orange}{#1}%
  \else%
    #2%
  \fi%
}\makeatother
```

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

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

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

```
\newcommand\phic{%
  \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.

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

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

```
\newcommand\phiConst{%
  \mathrel{\hspace{.15em}}%
  \mapstochar\mathrel{\hspace{-.15em}}\mapsto}
```

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

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

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

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

`\phiOset` Then, we define two commands to position a text over and under an arrow, as suggested here:

```
\makeatletter
\newcommand{\phiOset}[2]{%
  \mathrel{\mathop{#2}\limits^{
    \vbox to 0ex{\kern-2\ex@
    \hbox{$\scriptscriptstyle#1$}\vss}}}}
\newcommand{\phiUset}[2]{%
  \mathrel{\mathop{#2}\limits_{
    \vbox to 0ex{\kern-6.3\ex@
    \hbox{$\scriptscriptstyle#1$}\vss}}}}
\makeatother
```

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

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

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

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

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

```
\RequirePackage{trimclip}
\RequirePackage{amsfonts}
\makeatletter
\newcommand{\phiDotted}{%
  \mapstochar\mathrel{\mathpalette\phiDotted@\relax}}
\newcommand{\phiDotted@}[2]{%
  \begingroup%
  \settowidth{\dimen\z@}{$\m@th#1\rightarrow$}%
  \settoheight{\dimen\tw@}{$\m@th#1\rightarrow$}%
  \sbox\z@{%
    \makebox[\dimen\z@][s]{%
      \clipbox{0 0 {0.4\width} 0}%
        {\resizebox{\dimen\z@}{\height}%
          {$\m@th#1\dashrightarrow$}}%
      \hss%
      \clipbox{{0.69\width} {-0.1\height} 0
        {-\height}}{$\m@th#1\rightarrow$}%
    }%
  }%
  \ht\z@=\dimen\tw@ \dp\z@=\z@%
  \box\z@%
  \endgroup%
}
\makeatother
```