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

Yegor Bugayenko
yegor256@gmail.com

2022-12-16, 0.9.1

NB! You must run TeX processor with `--shell-escape` option and you must have Perl installed. This package doesn't work on Windows.

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 Bugayenko (2021) and later formalized by Kudasov et al. (2022). Here is how you render a simple expression:

$$
\begin{array}{l}
\texttt{app} \mapsto [\![\\
\quad \rho \mapsto \xi.b.\rho^2, \alpha_0 | t \rightsquigarrow \texttt{TRUE}, \\
\quad b \mapsto [\![\alpha_* \mapsto \texttt{fn}(\texttt{56}), \\
\quad\quad \varphi \mapsto \Phi.\texttt{hello.bye}(\xi), \\
\quad\quad \Delta \mapsto \texttt{01-FE-C3}]\!]]\!], \\
x \mapsto [\![\lambda \mapsto \varnothing]\!].
\end{array}
$$

```
\documentclass{article}
\pagestyle{empty}
\usepackage{eolang}
\begin{document}
\begin{phiquation*}
app -> [[ % it's abstract!
  ^ !-> $.b.^{^2}, 0/t~> TRUE,
  b -> [[ *-> fn(56),
    @ -> Q.hello.bye($),
    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:

*The sources are in GitHub at objectionary/eolang.sty

- "`@`" maps to "φ" (`\varphi`),
- "`^`" maps to "ρ" (`\rho`),
- "`$`" maps to "ξ" (`\xi`),
- "`&`" maps to "σ" (`\sigma`),
- "`?`" maps to "$\varnothing$" (`\varnothing`),
- "`Q`" maps to "Φ" (`\Phi`),
- "`->`" maps to "$\mapsto$" (`\mapsto`),
- "`~>`" maps to "$\rightsquigarrow$" (`\phiWave`),
- "`!->`" maps to "$\rightarrowtail$" (`\phiConst`),
- "`..>`" maps to "$\dashrightarrow$" (`\phiDotted`),
- "`D>`" maps to "$\Delta \dashrightarrow$" (`\Delta ..>`),
- "`L>`" maps to "$\lambda \dashrightarrow$" (`\lambda ..>`),
- "`[[`" maps to "$[\![$" (`\llbracket`),
- "`]]`" maps to "$]\!]$" (`\rrbracket`),
- "`|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 asterix 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 asterix 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.

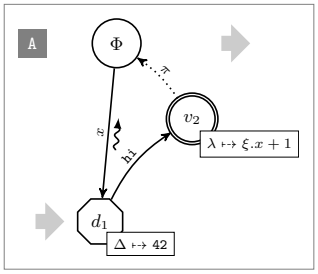
Texts in double quotes are automatically converted to fixed-width font too.

`\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 $y \mapsto [\![\Delta \dashrightarrow 42]\!]$.

```
\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:




```
\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.

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.

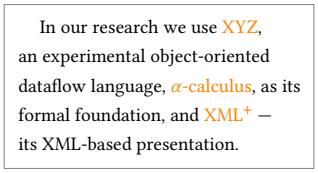
It is also possible to put transformation arrows to the graph, with the help of "`v0=>v1`" syntax. The arrow will be placed exactly between two vertices. You can also put an arrow from a vertex to the right, saying for example "`v3=>`", of from the left to the vertes, by saying for example "`=>v5`."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

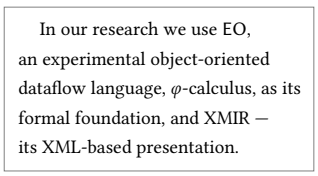
Be aware, unrecognized markers are simply ignored, without any error reporting.

`\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".



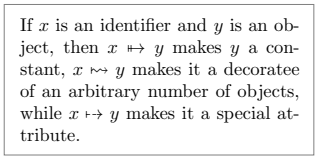

```
\usepackage[anonymous]{eolang}
\begin{document}
In our research we use \eolang{}, \\
an experimental object-oriented \\
dataflow language, \phic{}, as its \\
formal foundation, and \xmir{} --- \\
its XML-based presentation.
\end{document}
```

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



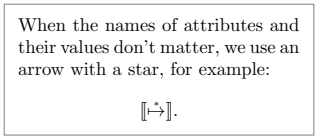

```
\usepackage{eolang}
\begin{document}
In our research we use \eolang{}, \\
an experimental object-oriented \\
dataflow language, \phic{}, as its \\
formal foundation, and \xmir{} --- \\
its XML-based presentation.
\end{document}
```

`\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 $x \phiConst y$
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:
\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 because of this):

The expression $[\![\alpha_1 \mapsto x_1, \alpha_2 \mapsto x_2, \ldots, \alpha_n \mapsto x_n]\!]$ 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:	$[\![x \mapsto [\![\Delta \mapsto 42]\!]]\!]$

```
\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:

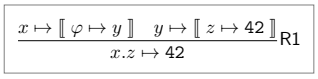
```
\usepackage[nodollar]{eolang}
```

`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}
```

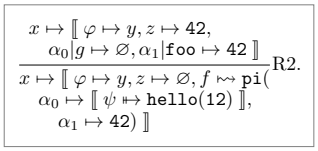
3 More Examples

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 the a continuation of the current line, you can use a single backslash as it's done here:



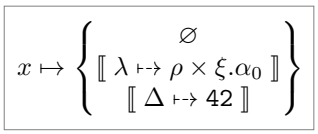
```
\begin{phiquation*}
\dfrac \
 {x->[[@->y]] \quad y->[[z->42]]} \
 {x.z -> 42} \
 \text{\sffamily R1}
\end{phiquation*}
```

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



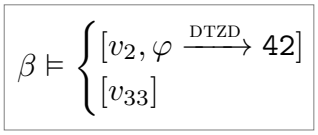
```
\begin{phiquation*}
\dfrac{\begin{split}
x->[[@->y, z->42,
  0/g->?, 1/foo->42]]
\end{split}}{\begin{split}
x->[[@->y, z->?, f ~> |pi|(
  0->[[ \psi !-> |hello|(12) ]],
    1->42)]]
\end{split}}\text{R2}.
\end{phiquation*}
```

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



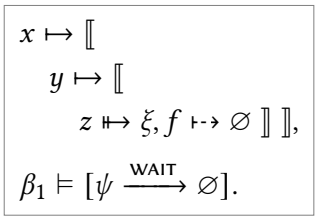
```
\begin{phiquation*}
x -> \left\{\begin{matrix} \
? \\
[[ L> ^ \times $.\alpha_0 ]] \\
[[ D> 42 ]] \
\end{matrix}\right\}
\end{phiquation*}
```

The `cases` environment works too:



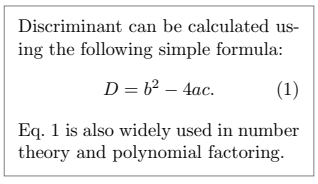
```
\begin{phiquation*}
\beta := \begin{cases} \
[ v_2, @ -dtzd> 42 ] \\
[ v_{33} ] \
\end{cases}
\end{phiquation*}
\end{document}
```

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



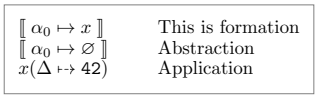
```
\documentclass{acmart}
\usepackage{eolang}
\thispagestyle{empty}
\begin{document}
\begin{phiquation*}
x -> [[
  y -> [[
    z !-> $, f ..> ? ]]]],\\
\beta_1 := [ \psi -wait> ? ].
\end{phiquation*}
\end{document}
```

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):



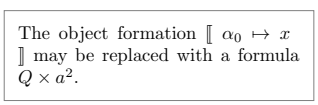
```
Discriminant can be calculated using
the following simple formula:
\begin{phiquation}
D = b{^2} - {4}ac.
\label{d}
\end{phiquation}
Eq.~\ref{d} is also widely used in
number theory and polynomial factoring.
```

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):



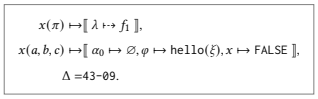
```
\begin{phiquation*}
[[ 0->x ]] && \text{This is formation}
[[ 0->? ]] && \text{Abstraction}
x(D>42) && \text{Application}
\end{phiquation*}
```

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 $[[0->x]]$
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:

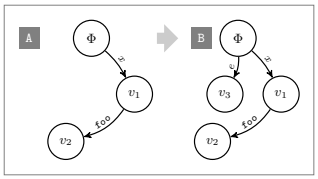


```
\begin{phiquation*}
x(\pi) -> [[\lambda ..> f_1]], \\
x(a,b,c) -> [[ \alpha_0 -> ?, \
  @ -> |hello|($), x -> |FALSE| ]], \\
\Delta = |43-09|.
\end{phiquation*}
```

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

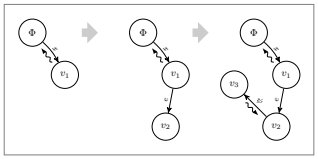
$$
\begin{phiquation*}
[[b -> ?]],
[[@ -> TRUE, \Delta ..> 42]], \\
\psi = << \pi, 42 >>.
\end{phiquation*}
$$

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



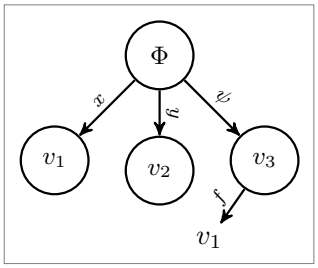
```
\begin{sodg}
v0 \\ v0!!A
v1 xy:v0,.7,1
v0->v1 a:x bend:-10
v2 xy:v1,-1.3,.8
v1->v2 a:|foo| bend:-20
v0+a xy:v0,3,0
v3a xy:v0a,-.7,1
v0a->v3a a:e bend:-15
v0=>v0a \\ v0a!B
\end{sodg}
```

You can make a copy from a copy:



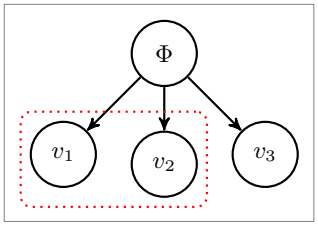
```
\begin{sodg}
v0
v1 xy:v0,.7,1
v0->v1 a:x bend:-10 rho
v0+a xy:v0,3,0 \\ v0=>v0a
v2a xy:v1a,-.8,1.3
v1a->v2a a:e
v0a+b xy:v0a,3,0 \\ v0a=>v0b
v3b xy:v2b,-1,-1
v2b->v3b a:\psi{} rho
\end{sodg}
```

You can have "broken" edges, using "`break`" attribute of an edge. The attribute must have a value, which is the percentange 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
v1 xy:v0,-1,1 \\ v0->v1
v2 xy:v0,0,1 \\ v0->v2
v3 xy:v0,1,1 \\ v0->v3
\node[draw=red,rounded corners,\
 dotted,fit=(v1) (v2)] {};
\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:

```
\RequirePackage{iexec}
```

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,
  tmpdir
}
\ProcessPgfPackageOptions{/eolang}
```

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

```
\iexec[null]{mkdir -p "\eolang@tmpdir/\jobname"}%
```

`\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
```

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

```
\makeatletter
\begin{VerbatimOut}{\eolang@tmpdir/eolang-phi.pl}
$macro = $ARGV[0];
open(my $fh, '<', $ARGV[1]);
my $tex; { local $/; $tex = <$fh>; }
print '% This file is auto-generated', "\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 $gathered = (0 == $tex =~ /\n\s+/g);
if ($gathered) {
print '% The "gathered" is used since all lines are left-aligned' . "\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') {
  $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;
$tex =~ s/(?<![{&])&(?![&])/\\sigma{}/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)(\]\[,.>\/]))([a-z][a-z0-9]+)(?=[\s)(\]\[,.-]|$)/|\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') {
  $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(?:->|:=|=)';
  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/\[\[/\\llbracket\\mathrel{}/g;
$tex =~ s/\]\]/\\mathrel{}\\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/!->/\\mathrel{\\phiConst}/g;
$tex =~ s/->/\\mathrel{\\mapsto}/g;
$tex =~ s/~>/\\mathrel{\\phiWave}/g;
$tex =~ s/:=/\\mathrel{\\vDash}/g;
$tex =~ s/\.\.>/\\mathrel{\\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 ($gathered) {
    print '\begin{gathered}';
  } else {
    print '\begin{split}';
  }
  print "\n";
  }
}
if ($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($align)) {
  print "\n";
  if ($gathered) {
    print '\end{gathered}';
  } else {
    print '\end{split}';
  }
  }
  print "\n" . '\end{' . $macro . '}';
}
print '\endinput';
\end{VerbatimOut}
\message{eolang: File with Perl script
  '\eolang@tmpdir/eolang-phi.pl' saved^^J}%
\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
```

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/phiquation.tex}}%
  \iexec[null]{cp "\eolang@tmpdir/\jobname/phiquation.tex"
    "\eolang@tmpdir/\jobname/\hash.tex"}%
  \message{Start parsing 'phi' at line no. \the\inputlineno^^J}
  \iexec[trace,stdout=\eolang@tmpdir/\jobname/\hash-post.tex]{
    perl "\eolang@tmpdir/eolang-phi.pl"
    '#1'
    "\eolang@tmpdir/\jobname/\hash.tex"
    \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/phiquation.tex}}
{\end{VerbatimOut}\eolang@process{equation*}}
\newenvironment{phiquation}%
{\catcode`\|=12 \VerbatimEnvironment%
\setcounter{eolang@lineno}{\value{FancyVerbLine}}%
\begin{VerbatimOut}
  {\eolang@tmpdir/\jobname/phiquation.tex}}
{\end{VerbatimOut}\eolang@process{equation}}
\makeatother
```

\phiq Then, we define \phiq command:

```
\RequirePackage{xstring}
\makeatletter\newcommand\phiq[1]{%
\StrSubstitute{\detokenize{#1}}{'}{'"'"'}[\clean]%
  \iexec[log,trace,quiet,stdout=\eolang@tmpdir/\jobname/phiq.tex]{
    /bin/echo '\clean'}%
  \def\hash{\eolang@mdfive
    {\eolang@tmpdir/\jobname/phiq.tex}}%
  \iexec[null]{cp "\eolang@tmpdir/\jobname/phiq.tex"
    "\eolang@tmpdir/\jobname/\hash.tex"}%
  \ifdefined\eolang@nodollar\else\catcode`\$=3 \fi%
  \iexec[trace,stdout=\eolang@tmpdir/\jobname/\hash-post.tex]{
    perl \eolang@tmpdir/eolang-phi.pl 'phiq'
    "\eolang@tmpdir/\jobname/\hash.tex"
    \ifdefined\eolang@nocomments | perl -pe 's/\%.*(\\n|$)//g'\fi}%
  \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
```

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

```
\makeatletter
\begin{VerbatimOut}{\eolang@tmpdir/eolang-sodg.pl}
sub num {
  my ($i) = @_;
  $i =~ s/(\+|-)\./\10./g;
  return $i;
}
sub fmt {
  my ($tex) = @_;
  $tex =~ s/\|([^\|]+)\|/\\textnormal{\\texttt{\1}}/g;
  return $tex;
}
sub vertex {
  my ($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 = {};
  foreach my $p (split(/ /, $tail)) {
    my ($q, $t) = split(/:/, $p);
    $opts{$q} = $t;
  }
  if (index($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 = $draw . ']';
    my ($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=.1cm,inner sep=0cm] ' .
        '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=' . ($size * 0.6) . 'cm of ' .
        $to . '.center]';
    } elsif ($to eq '') {
      print '\node [phi-arrow, right=' . ($size * 0.6) . 'cm 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 = () = $head =~ /!/g;
    print '\node [phi-marker, left=' .
      ($size * 0.6) . 'cm 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;
        $m =~ s/^(\s*)\Q${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 =~ /^v[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 ',' . $loc . '=';
      print abs(num($down)) . 'cm and ' .
        abs(num($right)) . 'cm of ' . $v . '.center';
    }
```

```
441     if (exists $opts{'data'}) {
442       print ',phi-data';
443       if ($opts{'data'} ne '') {
444         my $d = $opts{'data'};
445         if (index($d, '|') == -1) {
446           $d = '$\Delta\phiDotted\text{' .
447             '\textnormal{\texttt{' . fmt($d) . '}}}$';
448         } else {
449           $d = fmt($d);
450         }
451         $opts{'box'} = $d;
452       }
453     } elsif (exists $opts{'atom'}) {
454       print ',phi-atom';
455       if ($opts{'atom'} ne '') {
456         my $a = $opts{'atom'};
457         if (index($a, '$') == -1) {
458           $a = '$\lambda\phiDotted{}' . fmt($a) . '$';
459         } else {
460           $a = fmt($a);
461         }
462         $opts{'box'} = $a;
463       }
464     } else {
465       print ',phi-object';
466     }
467     print ']';
468     print ' (' . $head . ')';
469     print ' {';
470     if (exists $opts{'tag'}) {
471       my $t = $opts{'tag'};
472       if (index($t, '$') == -1) {
473         $t = '$' . $t . '$';
474       } else {
475        $t = fmt($t);
476      }
477      print $t;
478     } else {
479       print '$' . vertex($head) . '$';
480     }
481     print '}';
482     if (exists $opts{'box'}) {
483       print ' node[phi-box] at (';
484       print $head, '.south east) {';
485       print $opts{'box'}, '}';
486     }
487   } else {
488     print $cmd;
489   }
490   print ";\n";
491 }
492 print '\end{phicture}%', "\n";
493 print "% --- after processing:\n%";
494 foreach my $c (@cmds) {
```

```
  print '% ', $c, "\n";
}
print '% --- (' . (0+@cmds) . " lines)\n";
print '\endinput';
\end{VerbatimOut}
\message{eolang: File with Perl script
  '\eolang@tmpdir/eolang-sodg.pl' saved^^J}%
\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,thick,
    pics/parallel arrow/.style={
      code={\draw[-latex,phi-rho] (##1) -- (-##1);}}]}%
  {\end{tikzpicture}}
\tikzstyle{phi-arrow} = [fill=white!80!black, single arrow,
  minimum height=0.5cm, minimum width=0.5cm,
  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} = [thick,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,thin,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/sodg.tex}}
{\end{VerbatimOut}%
  \def\hash{\eolang@mdfive
    {\eolang@tmpdir/\jobname/sodg.tex}}%
  \iexec[null]{cp "\eolang@tmpdir/\jobname/sodg.tex"
    "\eolang@tmpdir/\jobname/\hash.tex"}%
  \catcode`\$=3 %
  \message{Start parsing `sodg' at line no. \the\inputlineno^^J}
  \iexec[trace,stdout=\eolang@tmpdir/\jobname/\hash-post.tex]{
    perl "\eolang@tmpdir/eolang-sodg.pl"
    "\eolang@tmpdir/\jobname/\hash.tex"
    \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
```

References

Bugayenko, Yegor (2021). *EOLANG and φ-calculus*. arXiv: 2111.13384 [cs.PL].
Kudasov, Nikolai et al. (2022). *φ-calculus: a purely object-oriented calculus of decorated objects*. arXiv: 2204.07454 [cs.PL].

Change History

Index

Numbers written in italic refer to the page where the corresponding entry is described; numbers underlined refer to the code line of the definition; numbers in roman refer to the code lines where the entry is used.

N

P

Q

R

S

T

U

V

W

X

Z