1 Introduction

This package helps you print formulas of $\varphi$-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{phiquation*}
app \rightarrow [ \\
\rho \mapsto \xi.b.\rho^2, \alpha_0/|t \mapsto \text{TRUE}, \\
b \mapsto [ \alpha \mapsto \text{fn}(56), \\
\varphi \mapsto \Phi\text{.hello.bye}(\xi), \\
\Delta \mapsto 01-FE-C3 ]], \\
x \mapsto [ \lambda \mapsto \emptyset ].
\end{phiquation*}

The environment \texttt{phiquation} lets you write a $\varphi$-calculus expressions using simple plain-text notation, where:

\begin{verbatim}
\documentclass{article} \\
pagestyle{empty} \\
\usepackage{eolang} \\
\begin{document} \\
\begin{phiquation*} \\
app \rightarrow [ \\
\rho \mapsto \xi.b.\rho^2, \alpha_0/|t \mapsto \text{TRUE}, \\
b \mapsto [ \alpha \mapsto \text{fn}(56), \\
\varphi \mapsto \Phi\text{.hello.bye}(\xi), \\
\Delta \mapsto 01-FE-C3 ]], \\
x \mapsto [ \lambda \mapsto \emptyset ].
\end{phiquation*} \\
\end{document}
\end{verbatim}

NB! You must run \TeX{} processor with \texttt{--shell-escape} option and you must have \texttt{Perl} installed. This package doesn’t work on Windows.

*The sources are in GitHub at objectionary/eolang.sty*
• “∅” maps to “φ” (\varphi),
• “−” maps to “ρ” (\rho),
• “$” maps to “ζ” (\zeta),
• “&” maps to “σ” (\sigma),
• “?” maps to “∅” (\varnothing),
• “Q” maps to “Φ” (\Phi),
• “->” maps to “↦” (\mapsto),
• “⇝” maps to “ϕWave”,
• “!->” maps to “ϕConst”,
• “..>” maps to “ϕDotted”,
• “D>” maps to “Δ..>” (\Delta ..>),
• “L>” maps to “λ..>” (\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),
• “>>” maps to “⟩” (\rangle),
• “-abc>” maps to “abc−→” (\phiSlot{abc}),
• “::=” maps to “⊨” (\vDash).

Before any arrow you can put a number, which will be rendered as \alpha with an index, for example \phi{0->x} will render “α0 \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 α0|g \mapsto x. You can use fixed-width words too, for example \phi{0/f->x} will render as “α0|f \mapsto x”. It’s also possible to use an asterix instead of a number, such that \phi{*/g->x} renders as “α*|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.

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

\begin{document}
\phiq{x \mapsto [\varphi \mapsto y]}
\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=>”, 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 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: \texttt{"v0+a"}. Here, we make a copy of \texttt{"v0"} and call it \texttt{"v0a"}. See the example below.

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

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

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

Without the anonymous option there will be no orange color:

\begin{verbatim}
\usepackage{eolang}
\begin{document}
In our research we use EO, an experimental object-oriented dataflow language, \phi-calculus, as its formal foundation, and XMIR --- its XML-based presentation.
\end{document}
\end{verbatim}

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

\begin{verbatim}
\phiConst\
\phiWave\
\phiDotted
\end{verbatim}

If \(x\) is an identifier and \(y\) is an object, then \(x \rightarrow y\) makes \(y\) a constant, \(x \Leftarrow y\) makes it a decoratee of an arbitrary number of objects, while \(x \Rightarrow y\) makes it a special attribute.

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

\begin{verbatim}
\phiOset
\phiUset
\end{verbatim}

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

\phiMany Sometimes you may need to simplify the way you describe an object (the typesetting
The expression $J_{\alpha_1 \rightarrow x_1, \alpha_2 \rightarrow x_2, \ldots, \alpha_n \rightarrow x_n}$ and expression $J_{\alpha_i \rightarrow i}$ are syntactically different but semantically equivalent.

Free: $[ x \rightarrow \emptyset ]$

Bound: $[ x \rightarrow \Delta \rightarrow 42 ]$

Free: & $[[ x \rightarrow ? ]]$

Bound: & \parbox{1in}{\input{a}} \par

\begin{phiquation*}
\phiq{[[ 1-> x_1, 2-> x_2, \dots, \alpha_n -> x_n ]]} and expression
\phiq{[[ \alpha_i \rightarrow \{i=1\}\{n\} x_i ]]} are syntactically different but
semantically equivalent.
\end{phiquation*}

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

\section{Package Options}

\texttt{ tmpdir } The default location of temp files is _eolang. You can change this with the help of the \texttt{tmpdir} package option:
\begin{verbatim}
usepackage[tmpdir=/tmp/foo]{eolang}
\end{verbatim}

\texttt{ nodollar } You may disable the special treatment of the dollar sign by using the \texttt{nodollar} package option:
\begin{verbatim}
usepackage[nodollar]{eolang}
\end{verbatim}

\texttt{ anonymous } You may anonymize \texttt{eolang}, \texttt{XMIR}, and \texttt{phic} commands by using anonymous package option (they all use the \texttt{eoAnon} command mentioned earlier):
\begin{verbatim}
usepackage[anonymous]{eolang}
\end{verbatim}
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*}
\begin{split}
x \rightarrow \begin{matrix}
\varphi & \rightarrow & y \\
y & \rightarrow & \begin{cases} 
\varphi & \rightarrow & 42 \\
\alpha_0 & \rightarrow & \varphi, \alpha_1 [f \rightarrow 42]
\end{cases}
\end{matrix}
z \rightarrow 42
\end{split}
\text{R1}
\end{phiquation*}

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

\begin{phiquation*}
\begin{split}
x \rightarrow \begin{matrix}
\varphi & \rightarrow & y, z \rightarrow 42, \\
\alpha_0 & \rightarrow & \varphi, \alpha_1 [f \rightarrow 42] \\
\alpha_1 & \rightarrow & \varphi [ \psi \rightarrow \text{hello}(12) ]
\end{matrix}
x \rightarrow \begin{cases} 
\alpha_0 & \rightarrow & \varphi, \alpha_1 [f \rightarrow 42] \\
\end{cases}
\end{split}
\text{R2}
\end{phiquation*}

You can use the \text{\textit{matrix}} environment too, in order to group a few lines:

\begin{phiquation*}
\begin{split}
x \rightarrow \begin{cases} 
\begin{matrix}
\varphi & \rightarrow & y, z \rightarrow 42, \\
\alpha_0 & \rightarrow & \varphi, \alpha_1 [f \rightarrow 42]
\end{matrix}
\end{cases}
\end{split}
\end{phiquation*}

The cases environment works too:

\begin{phiquation*}
\begin{cases} 
\begin{matrix}
\varphi & \rightarrow & y, z \rightarrow 42, \\
\alpha_0 & \rightarrow & \varphi, \alpha_1 [f \rightarrow 42]
\end{matrix}
\end{cases}
\end{phiquation*}

The phiquation environment may be used together with the \text{\textit{acmart}} package:
Discriminant can be calculated using the following simple formula:

\[
D = b^2 - 4ac. \tag{1}
\]

Eq. 1 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):

\[
\begin{phiquation*}
\begin{align*}
\text{This is formation} & \quad \text{Abstraction} \\
\text{Application} & \\
\end{align*}
\end{phiquation*}
\]

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:

\[
\begin{phiquation*}
\begin{align*}
\text{This is formation} & \quad \text{Abstraction} \\
\text{Application} & \\
\end{align*}
\end{phiquation*}
\]
If not a single line is indented in phiuation, all formulas will be centered:

\[ \begin{phiquation*}
[ b \mapsto \varnothing ],
[ \varphi \mapsto \text{TRUE, } \Delta \mapsto 42 ],
\psi = (\pi, 42).
\end{phiquation*} \]

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

\begin{code}
\begin{phiquation}
\begin{align*}
&v_0 \ \v_0!!A \\
&v_1 \ xy:v_0,.7,1 \\
&v_0\rightarrow v_1 \ a:x \ bend:-10 \\
&v_2 \ xy:v_1,-1.3,.8 \\
&v_1\rightarrow v_2 \ a:|foo| \ bend:-20 \\
&v_0+a \ xy:v_0,3,0 \\
&v_3a \ xy:v_0a,-.7,1 \\
&v_0a\rightarrow v_3a \ a:e \ bend:-15 \\
&v_0\Rightarrow v_0a \ \ \ \ \ \ \ \ \ v_0a!B
\end{align*}
\end{phiquation}
\end{code}

You can make a copy from a copy:

\begin{code}
\begin{phiquation}
\begin{align*}
&v_0 \\
&v_1 \ xy:v_0,.7,1 \\
&v_0\rightarrow v_1 \ a:x \ bend:-10 \ \text{rho} \\
&v_0+a \ xy:v_0,3,0 \ \ \ \ v_0\Rightarrow v_0a \\
&v_2a \ xy:v_1a,-.8,1.3 \\
&v_1a\rightarrow v_2a \ a:e \\
&v_0a+b \ xy:v_0a,3,0 \ \ v_0a\Rightarrow v_0b \\
&v_3b \ xy:v_2b,-1,-1 \\
&v_2b\rightarrow v_3b \ a:|\pi{}| \ \text{rho}
\end{align*}
\end{phiquation}
\end{code}

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{code}
\begin{phiquation}
\begin{align*}
&v_0 \\
&v_1 \ xy:v_0,-1,1 \\
&v_0\rightarrow v_1 \ a:x \\
&v_2 \ xy:v_0,0,1 \\
&v_0\rightarrow v_2 \ a:y \\
&v_3 \ xy:v_0,1,1 \\
&v_0\rightarrow v_3 \ a:|\pi{}| \\
&v_3\rightarrow v_1 \ a:f \ bend:-75 \ break:30
\end{align*}
\end{phiquation}
\end{code}
You can add TikZ commands to sodg graph, for example:
\begin{sodg}
\node(b){Φ};&\node(c){v_1};
\node(d){v_2};&\node(e){v_3};
&\draw[red, rounded corners, dotted, fit=(b) (d)]{v_0 \rightarrow v_1};
&\draw[red, rounded corners, dotted, fit=(b) (e)]{v_0 \rightarrow v_2};
&\draw[red, rounded corners, dotted, fit=(b) (e)]{v_0 \rightarrow v_3};
\end{sodg}

4 Implementation

First, we include a few packages. We need \texttt{stmaryrd} for \llbracket and \rrbracket commands:
\begin{verbatim}
\RequirePackage{stmaryrd}
\end{verbatim}

We need \texttt{amsmath} for equation* environment:
\begin{verbatim}
\RequirePackage{amsmath}
\end{verbatim}

We need \texttt{amssymb} for \varnothing command. We disable \texttt{Bbbk} because it may conflict with some packages from \texttt{acmart}:
\begin{verbatim}
\let\Bbbk\relax\RequirePackage{amssymb}
\end{verbatim}

We need \texttt{fancyvrb} for \VerbatimEnvironment command:
\begin{verbatim}
\RequirePackage{fancyvrb}
\end{verbatim}

We need \texttt{iexec} for executing Perl scripts:
\begin{verbatim}
\RequirePackage{iexec}
\end{verbatim}

Then, we process package options:
\begin{verbatim}
\RequirePackage{pgfopt}
\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}
\end{verbatim}

Then, we make a directory where all temporary files will be kept:
\begin{verbatim}
\iexec[null]{mkdir -p \"\eolang@tmpdir/\jobname\"}
\end{verbatim}

Then, we define an internal counter to protect line number from changing:
\begin{verbatim}
\makeatletter\newcounter{eolang@lineno}\makeatother
\end{verbatim}

Then, we define a command for MD5 hash calculating of a file:
\begin{verbatim}
\RequirePackage{pdftexcmds}
\makeatletter
\newcommand{\eolang@mdfive}[1]{\pdf@filemdfivesum{#1}}
\makeatother
\end{verbatim}
Then, we create a Perl script for phiquation processing using `VerbatimOut` environment from `fancyvrb`:

```perl
24 \makeatletter
25 \begin{VerbatimOut}{\eolang@tmpdir/eolang-phi.pl}
26 $macro = $ARGV[0];
27 open(my $fh, '<', $ARGV[1]);
28 my $tex; { local $/; $tex = <$fh>; }
29 print '% This file is auto-generated', "\n";
30 print '% There are ', length($tex), "\n";
31 ' chars in the input: ', $ARGV[1], "\n";
32 print '% ---', "\n";
33 if (index($tex, "\t") > 0) {
34 print "TABS are prohibited!";
35 exit 1;
36 }
37 my @lines = split (/\n/g, $tex);
38 foreach my $t (@lines) {
39 print '% ', $t, "\n";
40 }
41 print '% ---', "\n";
42 $tex =~ s/%.*\n/\n/g;
43 $tex =~ s/\n\s*//g;
44 my $gathered = (0 == $tex =~ /\n\s*/g);
45 if ($gathered) {
46 print '% The "gathered" is used since all lines are left-aligned'. "\n";
47 }
48 my $align = 0;
49 print '% The "align" is NOT used by default'. "\n";
50 if (index($tex, "&&") >= 0) {
51 $macro =~ s/equation/align/g;
52 $align = 1;
53 print '% The "align" is used because of && seen in the text'. "\n";
54 }
55 if ($macro ne 'phiq') {
56 $tex =~ s/\n\n\n\n/\n/g;
57 $tex =~ s/\n\n\n\n\n/\n/g;
58 $tex =~ s/\n\n\n{label\{-[\}\]+}}\n*/\1/g;
59 $tex =~ s/\n\n(3,)/\n/g;
60 }
61 my @texts = ();
62 sub trep {  
63 my ($s) = @_;  
64 my $open = 0;  
65 my $p = 0;  
66 for (; $p < length($s); $p++) {
67 $c = substr($s, $p, 1);  
68 if ($c eq '}') {  
69 if ($open eq 0) {
70 last;
71 }
72 $open--;  
73 }
74 if ($c eq '{') {
75 $open++;  
76 }  
77 }  
78 \end{VerbatimOut}
79 \end{VerbatimOut}
80 \end{VerbatimOut}
"eolang-phi.pl"
push(@texts, substr($s, 0, $p));
return 'TEXT' . (0+@texts - 1) . '}' . substr($s, $p + 1);

$tex =~ s/\text{(.+?)/trep("$1")/g;
$tex =~ s/\begin{split}
/\begin{split}&/g;
$tex =~ s/\end{split}/\end{split}/g;
$tex =~ s/\$//g;
$tex =~ s/\//\quad{}/g;
$tex =~ s/\(/\langle/;
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:

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

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:

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

```latex
\makeatletter
\newcommand\phiSaveTo[1]{\def\eolang@phiSaveTo{#1}}
\makeatother
```
Then, we define the phiquation and the phiquation* environments through a supplementary \eolang@process command:

\makeatletter\newcommand\eolang@process[1]{
  \def\hash{\eolang@mdfive
    \jobname/phiquation.*}
  \iexec[null]{cp "\jobname/phiquation.*"
    "\jobname/\hash.*"}
  \message{Start parsing 'phi' at line no. \the\inputlineno}
  \iexec[trace,stdout=\jobname/\hash-post.*]{perl "\jobname/eolang-phi.pl"
    '#1'}
  \iexec[log,trace,quiet,stdout=\jobname/\hash-post.*]{perl -pe 's/\%.*(\n|$)/g'}
  \message{Start parsing 'phi' at line no. \the\inputlineno}
  \iexec[trace,stdout=\jobname/\hash-post.*]{perl -pe 's/\%.*(\n|$)/g'}
  \ifdefined\eolang@nodollar | perl -pe 's/\%.*(\n|$)/g'} fi
  \message{Start parsing 'phi' at line no. \the\inputlineno}
  \iexec[trace,stdout=\jobname/\hash-post.*]{perl -pe 's/\%.*(\n|$)/g'}
  \ifdefined\eolang@nodollar | perl -pe 's/\%.*(\n|$)/g'} fi
}
\newenvironment{phiquation*}{}{
  \VerbatimEnvironment
  \begin{VerbatimOut}
    \jobname/phiquation.*}
  \end{VerbatimOut}
  \eolang@process{equation*}}
\newenvironment{phiquation}{}{
  \VerbatimEnvironment
  \begin{VerbatimOut}
    \jobname/phiquation.*}
  \end{VerbatimOut}
  \eolang@process{equation}}
\makeatother

\phiq Then, we define \phiq command:

\RequirePackage{xstring}
\makeatletter\newcommand\phiq[1]{
  \StrSubstitute{\detokenize{#1}}{"\textquotesingle"}{\textquotedbl}'\clean
  \iexec[log,trace,quiet,stdout=\jobname/\phiq.*]{
    /bin/echo '\clean'}
  \def\hash{\eolang@mdfive
    \jobname/\phiq.*}
  \iexec[null]{cp "\jobname/\phiq.*"
    "\jobname/\hash.*"}
  \message{Start parsing 'phi' at line no. \the\inputlineno}
  \iexec[trace,stdout=\jobname/\hash-post.*]{perl "\jobname/eolang-phi.pl"
    '#1'}
  \iexec[log,trace,quiet,stdout=\jobname/\hash-post.*]{perl -pe 's/\%.*(\n|$)/g'}
  \ifdefined\eolang@nodollar | perl -pe 's/\%.*(\n|$)/g'} fi
  \message{Start parsing 'phi' at line no. \the\inputlineno}
  \iexec[log,trace,quiet,stdout=\jobname/\hash-post.*]{perl -pe 's/\%.*(\n|$)/g'}
  \ifdefined\eolang@nodollar | perl -pe 's/\%.*(\n|$)/g'} fi
  \makeatother

\nodollar Then, we redefine dollar sign:

\ifdefined\eolang@nodollar \else \begingroup
\begin{VerbatimOut}
\end{VerbatimOut}
\endgroup
Then, we create a Perl script for sodg graphs processing using \texttt{VerbatimOut} from \texttt{fancyvrb}:

\begin{VerbatimOut}{/eolang@tmpdir/eolang-sodg.pl}

\begin{verbatim}
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\n"; 
}
print "% ---\n\n"; 
$tex =~ s/\n/\n\n/g; 
$tex =~ s/\n\n/g; 
$tex =~ s/\s+/\s+/g; 
$tex =~ s/\s\n/g; 
my @cmds = split (\n/g, $tex); 
\end{verbatim}
\end{VerbatimOut}

\eolang-sodg.pl
print '" --- before processing:' . "\n";
foreach my $t (split (/\n/g, $tex)) {
  print '" ', $t, "\n";
}
print '" ---';
print ('(' . (0+@cmds) . " lines)\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[1';
    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 . '\draw
    } else {
      $draw = $draw . '->\draw
    } else {
      my $a = $opts{'a'};
      if (index($a, '$') == -1) {
        $a = '$' . fmt($a) . '$';
      }
      $draw = $draw . '<MB:node [phi-attr] {' . $a . '}>';}
    if (exists $opts{'break'}) {
      $draw = $draw . '<F:coordinate [pos=\f' . ($opts{'break'} / 100) . '] (break)>';
    } else {
      $draw = $draw . '->\draw
    }
$draw = \$draw . " (<MF:${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${suffix}\E\s?/#\Q${h}\E\${suffix} /g) {
            last;
        }
        $m =~ s/^\s*\Q${f}\E\s/#\Q${h}\E\${suffix} /g;
        $m =~ s/^\s*\Q${f}\E->/#\Q${h}\E->/g;
        $m =~ s/\sxy:\Q${f}\E,/ xy:\Q${h}\E/suffix\G/;
        $m =~ s/^\s*->\Q${f}\E\s/#\Q${h}\E->/g;
        if ($m ne $e) {
            push(@extra, ' ' . $m);
        }
    }
    splice(@extra, 0, 0, @extra[-1]);
    splice(@extra, -1, 1);
    splice(@extra, 0, 0, ' % clone of ' . $v . ', 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 ';
        }
        print ',' . $loc . '=
        abs(num($down)) . ' cm and ' . abs(num($right)) . ' cm of ' . $v . '.center';
    }
if (exists $opts{'data'}) {
    print ',phi-data';
    if ($opts{'data'} ne '') {
        my $d = $opts{'data'};
        if (index($d, '!') == -1) {
            $d = '\Delta\phiDotted\text{'} . 'texttt{' . fmt($d) . '}\$';
        } else {
            $d = fmt($d);
        }
        $opts{'box'} = $d;
    }
    if (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';
        }
    } else {
        print ',phi-object';
    }
    print '];
    print '\text{'} . $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 '};
}
else {
    print ',\text{'} . $head . '. ');
    print '{';
    if (exists $opts{'box'}) {
        print ' node[phi-box] at (';
        print $head, '. south east) {';
        print $opts{'box'}, '}';
    } else {
        print $cmd;
    }
    print ',\n';
}

FancyVerbLine Then, we reset the counter for fancyverb, so that it starts counting lines from zero when
the document starts rendering:
\setcounter{FancyVerbLine}{0}

\begin{VerbatimOut}

\message{eolang: File with Perl script \eolang@tmpdir/eolang-sodg.pl' saved^^J}%

\makeatother

\begin{VerbatimOut}

\message{eolang: File with Perl script \eolang@tmpdir/eolang-sodg.pl' saved^^J}%

\makeatother

\end{VerbatimOut}

\endinput

...
\v = ($\langle I \rangle - \langle F \rangle$);
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
}]\n{$\langle F \rangle!.5!(\langle I \rangle + (\t: -\d em) + (\t + \a: 1ex)$}\n-- ++(\t: 2*\d em);
};
}
lineto code={
\tikzmath{
coordinate \ I, \ F, \ v;
\ I = (\tikzinputsegmentfirst);
\ F = (\tikzinputsegmentlast);
\ v = ($\langle I \rangle - \langle F \rangle$);
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
}]\n{$\langle F \rangle!.5!(\langle I \rangle + (\t: -\d em) + (\t + \a: 1ex)$}\n-- ++(\t: 2*\d em);
};
}
\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.
Then, we create a new environment `sodg`, as suggested here:

```latex
\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"
\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
```

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

```latex
\newcommand\eoAnon[O{ANONYMIZED}m]{\ifdefined\eolang@anonymous\textcolor{orange}{#1}\else#2\fi}
```

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

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

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

```latex
\newcommand\phic{\eoAnon[\(\alpha\)-calculus]{\(\varphi\)-calculus}}
```

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

\eolang\eoAnon[\{\alpha\}-cal\-cu\-lus]\{\{\varphi\i-cal\-cu\-lus\}

\phic Then, we define a simple supplementary command to help you print $\varphi$-calculus, the name of our formal apparatus.
Then, we define a simple supplementary command to help you print XMIR, the name of our XML-based format of program representation.

\newcommand\xmir{\texttt{Anon}[XML\(+\)](XMIR)}

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

\newcommand\phiConst{\mapstochar\mapsto}

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}

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

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

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

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

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

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

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


Change History

0.0.1
General: First draft. .......................... 9

0.0.2
sodg: The environment `phigure` renamed to `sodg` for the sake of better semantic. The graph in the picture is solely a SODG graph, that’s why the name `sodg` is better. 21
eolang-phi.pl: New symbol added for basket slots .......................... 10
Parsing of the symbols “@”, “ˆ”, and “&” enabled (\varphi, \rho, and \sigma) .......................... 10
The symbols “[” and “]” replaced with “[[” and “]]” for abstract object brackets, because they conflicted with normal square brackets .......................... 10
eolang-sodg.pl: The Perl file now has a fixed name, which doesn’t depend on the name of the TeX job. This file may be shared among jobs, no need to make it uniquely named. 14
\phiq: Parsing of additional symbols enabled. .......................... 13

0.1.0
General: Parsing of package options introduced. .......................... 9
\eolang: New command \eolang added to print the name of the language in both normal and the anonymous mode of \acmart. .......................... 21
\eolang@mdfive: New supplementary command added to calculate MD5 sum of a file. .......................... 9
eolang-phi.pl: A new Perl script “eolang-phi.pl” added for parsing of phi expressions. .......................... 10
eolang-sodg.pl: There are two Perl scripts now: one for phiquation, another one for sodg. .......................... 14
\phic: New command \phic prints the name of ϕ-calculus in both normal and the anonymous mode of \acmart. .......................... 21
\phiConst: New command \phiConst added to denote a link to a constant attribute. .......................... 22
\phiDotted: New command \phiDotted added to denote a link to a special attribute. .......................... 22

0.2.0
eolang-phi.pl: Numbers automatically render as \texttt
No need to use vertical bars around them anymore. .......................... 10
eolang-sodg.pl: The content of the atom and the data boxes is parsed automatically as formulas and numbers, respectively. .......................... 14
\xmir: New command \xmir prints XMIR in both normal and the anonymous mode of \acmart. .......................... 22

0.3.0
\eolang@lineno: New counter for protecting lineno. .......................... 9
\phicWave: New command \phicWave added to denote a link to a multi-layer attribute. .......................... 22

0.4.0
eolang-sodg.pl: Labels on the edges are automatically printed as math formulas. Also, boxes are prefixed with the \Delta and the \lambda commands. .......................... 14
Relative positioning of vertices fixed. .......................... 14

0.5.0
eolang-phi.pl: Automated formatting of TRUE and FALSE added. .......................... 10
eolang-sodg.pl: It is possible to use TikZ commands inside the sodg environment. .......................... 14
New syntax introduced that allows to make clones of vertices and all their dependants. .......................... 14
Now edges may have the break attribute, to make them shorter. .......................... 14
\phiMany: New command \phiMany enables iterating over an arrow. .......................... 22
\phiSlot: New command \phiSlot added to denote a link to a slot in a basket. .......................... 22

0.6.0
General: Package option nocomments added in order to enable comments suppression in temporary .tex files (may be pretty important for .dtx documents). .......................... 9

25
eolang-sodg.pl: The rrho attribute is retired, now rho works just fine in all situations. ............... 14

0.7.0

nodollar: Now it is possible to use dollar sign instead of the \phiq command. ............... 13

eolang-phi.pl: New syntax sugar for \Phi, just using capital “Q” is enough. 10

Object names are automatically converted to \texttt, provided their names include two or more symbols. ............... 10

Text in quotes is automatically converted to \texttt. ............... 10

0.8.0

General: The anonymous package option added. ............... 9

eolang-phi.pl: Inside phiuation any text inside the \text macro is not processed. ............... 10

eolang-sodg.pl: The tag attribute is introduced for changing labels inside a vertex circle. ............... 14

\phiOset: New commands \phiOset and \phiUset help position text over and under an arrow. ............... 22

\phiSaveTo: The output of the phiuation environment can be redirected to a file. ............... 12

\sodgSaveTo: The output of the sodg environment can be redirected to a file. ............... 20

0.9.0

\eoAnon: New command \eoAnon added. ............... 21

eolang-phi.pl: Proper handling of the matrix environment. ............... 10

\phiEOL: New command \phiEOL added, instead of \\[-4pt\]. ............... 22
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.

Symbols

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Code Lines</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>122, 219, 224, 228, 231, 606, 613</td>
</tr>
<tr>
<td>%</td>
<td>192, 223, 611</td>
</tr>
<tr>
<td>(</td>
<td>619, 635, 637</td>
</tr>
<tr>
<td>)</td>
<td>620, 635, 637</td>
</tr>
<tr>
<td>*</td>
<td>88, 90, 119</td>
</tr>
<tr>
<td>+</td>
<td>237, 364, 395, 401</td>
</tr>
<tr>
<td>-</td>
<td>89, 91, 127, 137, 237</td>
</tr>
<tr>
<td>/</td>
<td>87, 88</td>
</tr>
<tr>
<td>?</td>
<td>130</td>
</tr>
<tr>
<td>[</td>
<td>87, 124</td>
</tr>
<tr>
<td>{</td>
<td>58, 81, 93, 94, 119, 123, 127, 142</td>
</tr>
<tr>
<td>}</td>
<td>58, 93, 94, 119, 142</td>
</tr>
<tr>
<td>]</td>
<td>87, 125</td>
</tr>
<tr>
<td>|</td>
<td>123</td>
</tr>
<tr>
<td>\l</td>
<td>88, 140, 141, 198, 204, 242, 597</td>
</tr>
</tbody>
</table>


