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

Yegor Bugayenko
yegor256@gmail.com

2022-10-25, 0.0.2

NB! You must run TeX processor with `--shell-escape` option and you must have Perl installed.

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:

$$
\begin{aligned}
&a \mapsto \llbracket \\
&\quad \rho \Mapsto \xi.b, \\
&\quad b \mapsto \llbracket c \mapsto \texttt{fn}(56), \\
&\qquad \varphi \mapsto \texttt{hello}(\xi), \\
&\qquad \Delta \dashrightarrow \texttt{01-FE-C3} \rrbracket \rrbracket, \\
&x \mapsto \llbracket \alpha_0 \mapsto \varnothing \rrbracket
\end{aligned}
$$

```
\documentclass{article}
\pagestyle{empty}
\usepackage{eolang}
\begin{document}
\begin{phiquation*}
a -> [[
  ^ !-> $.b,
  b -> [[ c -> |fn|(56),
    @ -> |hello|($),
    \Delta ~> |01-FE-C3| ]]]],

x -> [[ \alpha_0 -> ? ]]
\end{phiquation*}
\end{document}
```

`phiquation` (*env.*) The environment `phiquation` lets you write a φ-calculus expressions using simple

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

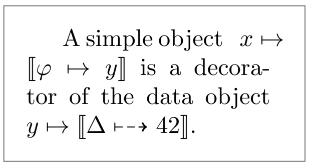
plain-text notation, where:

- "`@`" maps to "φ" (`\varphi`),
- "`^`" maps to "ρ" (`\rho`),
- "`$`" maps to "ξ" (`\xi`),
- "`&`" maps to "σ" (`\sigma`),
- "`?`" maps to "$\varnothing$" (`\varnothing`),
- "`->`" maps to "$\mapsto$" (`\mapsto`),
- "`!->`" maps to "$\mapstochar\relbar\mathrel{\mkern-12mu}\mapsto$" (`\mapstochar\relbar\mathrel{\mkern-12mu}\mapsto`),
- "`~>`" maps to "$\mapstochar\dashrightarrow$" (`\mapstochar\dashrightarrow`),
- "`[[`" maps to "$\llbracket$" (`\llbracket`),
- "`]]`" maps to "$\rrbracket$" (`\rrbracket`),
- "`|abc|`" maps to "`abc`" (`\texttt{abc}`).

Also, a few symbols are supported for φPU architecture:

- "`-abc>`" maps to "$\xrightarrow{\text{ABC}}$" (`\xrightarrow{\text{\sffamily\scshape abc}}`),
- "`:=`" maps to "$\vDash$" (`\vDash`).

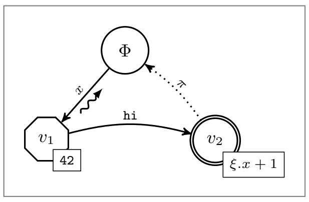
`\phiq` The command `\phiq` lets you inline a φ-calculus expressions using the same simple plain-text notation:



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

```
\documentclass{article}
\usepackage[paperwidth=2in]{geometry}
\pagestyle{empty}
\usepackage{eolang}
\begin{document}
A simple object
\phiq{x -> [[@ -> y]]}
is a decorator of
the data object
\phiq{y -> [[\Delta ~> 42]]}.
\end{document}
```

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



```
\documentclass{article}
\pagestyle{empty}
\usepackage{eolang}
\begin{document}
\begin{sodg}
v0
v1 xy:v0,-2,+1 data:|42|
v0->v1 a:$x$ rho
v2 xy:v0,+1,+1 atom:$\xi.x+1$
v1->v2 a:|hi| bend:-15
v2->v0 pi bend:10
\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:

- “`data:[<box>]`” makes it a data vertex with an optional attached `<box>`,
- “`atom:[<box>]`” makes it an atom with an optional attached `<box>`,
- “`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.

The following markers are supported for an edge:

- “`rho`” places a backward snake arrow to the edge,
- “`rrho`” places a reverse `rho`,
- “`bend:<angle>`” bend it right by the amount of `<angle>`,
- “`a:<txt>`” attaches label `<txt>` to it,
- “`pi`” makes it dotted, with π label.

2 Implementation

First, we include a few packages:

```
\RequirePackage{stmaryrd}
\RequirePackage{amsmath}
\RequirePackage{amssymb}
\RequirePackage{amsfonts}
\RequirePackage{iexec}
\RequirePackage{fancyvrb}
```

`\eolang@env` Then, we define `\eolang@env` supplementary command. It is implemented with the help of `\iexec` from iexec package:

```
\makeatletter\newcommand\eolang@env[2]{
  \iexec[trace]{
    /bin/echo -n ’\\begin{#1}\\begin{split} &’
    &&
    /bin/echo -n ’\detokenize{#2}’
      | perl -pe ’s/^\\r\\+//g’
      | perl -pe ’s/\\r\\+$//g’
      | perl -pe ’s/\\?/\\\\varnothing/g’
      | perl -pe ’s/@/\\\\varphi/g’
      | perl -pe ’s/&/\\\\sigma/g’
      | perl -pe ’s/\\^/\\\\rho/g’
      | perl -pe ’s/\\$/\\\\xi/g’
      | perl -pe ’s/-([a-z]+)>/\\\\xrightarrow{\\\\text{\\\\sffamily\\\\scshape \\1}}/g’
      | perl -pe ’s/!->/\\\\mapstochar\\\\relbar\\\\mathrel{\\\\mkern-12mu}\\\\mapsto/g’
      | perl -pe ’s/->/\\\\mapsto/g’
      | perl -pe ’s/:=/\\\\vDash/g’
      | perl -pe ’s/\unexpanded{~}>/\\\\mapstochar\\\\dashrightarrow/g’
      | perl -pe ’s/\\|([^\\|]+)\\|/\\\\texttt{\\1}/g’
      | perl -pe ’s/\\[\\[/\\\\llbracket/g’
      | perl -pe ’s/\\]\\]/\\\\rrbracket/g’
      | perl -pe ’s/\\r\\r/\\\\\\\\ \&/g’
      | perl -pe ’s/\\r/\\\\\\\\[-4pt] \&/g’
      | perl -pe ’s/([^& ]) {2}([^ ])/\\1 \\2/g’
      | perl -pe ’s/ {2}/\\\\quad{}/g’
    &&
    /bin/echo -n ’\\end{split}\\end{#1}\\endinput’
  }%
}\makeatother
```

phiquation Then, we define phiquation and phiquation* environments:

```
\makeatletter
\NewDocumentEnvironment{phiquation*}{b}{%
  \eolang@env{equation*}{#1}
}{}
\NewDocumentEnvironment{phiquation}{b}{%
  \eolang@env{equation}{#1}
}{}
\makeatother
\AddToHook{env/phiquation*/begin}{\obeylines\obeyspaces}
\AddToHook{env/phiquation/begin}{\obeylines\obeyspaces}
```

\phiq Then, we define \phiq command:

```
\newcommand\phiq[1]{
  \iexec[trace]{
    /bin/echo -n ’$’
    &&
    /bin/echo -n ’\detokenize{#1}’
      | perl -pe ’s/\\^/\\\\rho/g’
      | perl -pe ’s/\\$/\\\\xi/g’
      | perl -pe ’s/&/\\\\sigma/g’
      | perl -pe ’s/\\?/\\\\varnothing/g’
      | perl -pe ’s/@/\\\\varphi/g’
      | perl -pe ’s/!->/\\\\mapstochar\\\\relbar\\\\mathrel{\\\\mkern-12mu}\\\\mapsto/g’
      | perl -pe ’s/->/\\\\mapsto/g’
      | perl -pe ’s/-([a-z]+)>/\\\\xrightarrow{\\\\text{\\\\sffamily\\\\scshape \\1}}/g’
      | perl -pe ’s/\unexpanded{~}>/\\\\mapstochar\\\\dashrightarrow/g’
      | perl -pe ’s/:=/\\\\vDash/g’
      | perl -pe ’s/\\[\\[/\\\\llbracket/g’
      | perl -pe ’s/\\]\\]/\\\\rrbracket/g’
    &&
    /bin/echo -n ’$\\endinput’
  }%
}
```

Perl Then, create a Perl script:

```
\begin{VerbatimOut}{eolang.pl}
$tex = $ARGV[0];
$tex =~ s/^\s+|\s+$//g;
$tex =~ s/(\\[a-zA-Z]+)\s+/\1/g;
$tex =~ s/\r\s+/\r/g;
$tex =~ s/\|([^\|]+)\|/\\texttt{\1}/g;
my @cmds = split (/\r/g, $tex);
```

```
print '\begin{phig}', "\n";
foreach my $c (@cmds) {
  my ($head, $tail) = split (/ /, $c, 2);
  my %opts = {};
  foreach my $p (split (/ /, $tail)) {
    my ($q, $t) = split (/:/, $p);
    $opts{$q} = $t;
  }
  if (index($head, '->') == -1) {
    print '\node[';
    if (exists $opts{'xy'}) {
      my ($v, $right, $down) = split(/,/, $opts{'xy'});
      print ',below right=';
      print $down;
      print 'cm and ';
      print $right;
      print 'cm of ';
      print $v;
    }
    if (exists $opts{'data'}) {
      print ',phi-data';
      if (not $opts{'data'} eq '') {
        $opts{'box'} = $opts{'data'};
      }
    } elsif (exists $opts{'atom'}) {
      print ',phi-atom';
      if (not $opts{'atom'} eq '') {
        $opts{'box'} = $opts{'atom'};
      }
    } else {
      print ',phi-object';
    }
    print ']';
    print ' (', $head, ')';
    print ' {$';
    if ($head eq 'v0') {
      print '\Phi';
    } else {
      print 'v_', substr($head, 1);
    }
    print '$}';
    if (exists $opts{'box'}) {
      print ' node[phi-box] at (';
      print $head, '.south east) {';
      print $opts{'box'}, '}';
    }
  } else {
    print '\draw[';
    if (exists $opts{'pi'}) {
      print ',phi-pi';
      if (not exists $opts{'a'}) {
        $opts{'a'} = '$\pi$';
      }
    }
    print ']';
    my ($from, $to) = split (/->/, $head);
    print ' (', $from, ') ';
    if (exists $opts{'bend'}) {
      print 'edge [bend right=', $opts{'bend'}, ']';
    } else {
      print '--';
    }
    if (exists $opts{'rho'} or exists $opts{'rrho'}) {
      print ' pic[sloped,phi-rho]{parallel arrow={';
      print '-' if not exists $opts{'rrho'};
      print '0.3,-0.15}}';
    }
    if (exists $opts{'a'}) {
      print ' node [phi-attr] {', $opts{'a'}, '}';
    }
    print ' (', $to, ')';
  }
  print ";\n";
}
print '\end{phig}', "\n", '\endinput';
\end{VerbatimOut}
\message{^^Jeolang: File with Perl script (eolang.pl) saved^^J}%
\iexec[trace,null]{perl -pi -e 's/(\\\\[a-zA-Z])\\s+/\\1/g' eolang.pl}
```

tikz Then, we include tikz package and its libraries:

```
\RequirePackage{tikz}
  \usetikzlibrary{arrows}
  \usetikzlibrary{shapes}
  \usetikzlibrary{decorations}
  \usetikzlibrary{decorations.pathmorphing}
  \usetikzlibrary{intersections}
  \usetikzlibrary{positioning}
  \usetikzlibrary{calc}
  \usetikzlibrary{shapes.arrows}
```

phig Then, we define internal environment phig:

```
\newenvironment{phig}%
  {\noindent\begin{tikzpicture}[
    ->,>=stealth',node distance=0,thick,
    pics/parallel arrow/.style={
      code={\draw[-latex,phi-rho] (##1) -- (-##1);}}]}%
  {\end{tikzpicture}}
\tikzstyle{transforms} = [fill=white!80!black, single arrow,
  minimum height=0.5cm, minimum width=0.5cm,
  single arrow head extend=2mm]
\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]
\tikzstyle{phi-rho} = [draw,decorate,decoration={
  snake,amplitude=.4mm,segment length=2mm,post length=1mm}]
```

```
\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]
```

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

```
\NewDocumentEnvironment{sodg}{b}{%
  \catcode`\ =10 %
  \catcode`\^^M=5 %
  \iexec[trace,stdout=\jobname.tex.eolang]{
    perl eolang.pl '\detokenize{#1}'}%
}{}
\AddToHook{env/sodg/before}{\bgroup\obeylines\obeyspaces}
\AddToHook{env/sodg/after}{\egroup}
```

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

0.0.2

`\eolang@env`: New symbol added for basket slots . . . 3

Parsing of symbols “@,” “^,” and “&” enabled (varphi, rho, and sigma) . . 3

The symbols “[” and “]” replaced with “[[” and “]]” for abstract object brackets, because they conflicted with normal square brackets . . . 3

`Perl`: 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. . . 4

`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. . . . 7

`\phiq`: Parsing of additional symbols enabled . . . 4

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.