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

Yegor Bugayenko
`yegor256@gmail.com`

2022-10-29, 0.2.0

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:

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

```
\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 plain-text notation, where:

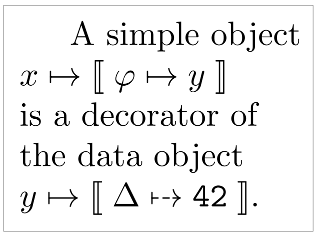
[*] The sources are in GitHub at objectionary/eolang.sty

- “`@`” maps to “φ” (`\varphi`),
- “`^`” maps to “ρ” (`\rho`),
- “`$`” maps to “ξ” (`\xi`),
- “`&`” maps to “σ” (`\sigma`),
- “`?`” maps to “$\varnothing$” (`\varnothing`),
- “`->`” maps to “$\mapsto$” (`\mapsto`),
- “`!->`” maps to “$\Mapsto$” (`\phiConst`),
- “`..>`” maps to “$\dashrightarrow$” (`\phiDotted`),
- “`[[`” maps to “$\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`).

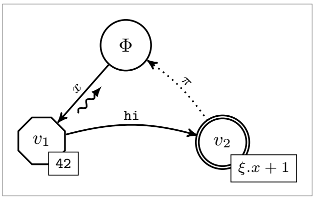
`\phiq` The command `\phiq` lets you inline a φ-calculus expressions using the same simple plain-text notation:



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

```
\documentclass{article}
\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}
```

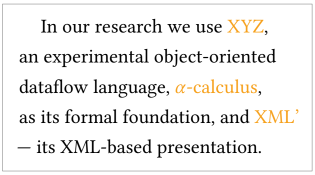
The content of the environment is parsed line by line. Markers in each line are separated by a single space. The first marker is either a unique name of a vertex, like `v1` in the example above, or an edge, like `v0->v1`. All other markers are either unary like `rho` or binary like `atom:$\xi.x+1$`. 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>` (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.

The following markers are supported for an edge:

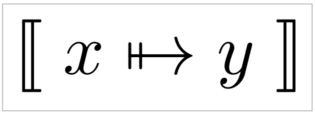
- "`rho`" places a backward snake arrow to the edge,
- "`rrho`" places a reverse `rho`,
- "`bend:<angle>`" bend it right by the amount of `<angle>`,
- "`a:<txt>`" attaches label `<txt>` to it,
- "`pi`" makes it dotted, with π label.

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


In our research we use XYZ,
an experimental object-oriented
dataflow language, α-calculus,
as its formal foundation, and XML'
— its XML-based presentation.

```
\documentclass[anonymous]{acmart}
\thispagestyle{empty}
\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` A simple commands is defined to help you render an arrow for a constant attribute. It is recommende not to use it directly, but use `!->` instead. However, if you want to use `\phiConst`, wrap it in `\mathrel` for better display:


$$[\![\; x \mapsto y \;]\!]$$

```
\documentclass{article}
\pagestyle{empty}
\usepackage{eolang}
\begin{document}
\phiq{[[ x \mathrel{\phiConst} y ]]}
\end{document}
```

2 Package Options

`tmpdir` The default location of temp files is `_eolang`. You can change this using `tmpdir` option:

```
\usepackage[tmpdir=/tmp/foo]{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:

$$\dfrac{x \mapsto \llbracket\, \varphi \mapsto y \,\rrbracket \quad y \mapsto \llbracket\, z \mapsto 42 \,\rrbracket}{x.z \mapsto 42}\textsf{R1}$$

```
\documentclass{article}
\usepackage{amsmath}
\usepackage{eolang}
\pagestyle{empty}
\begin{document}
\begin{phiquation*}
\dfrac \
 {x->[[@->y]] \quad y->[[z->42]]} \
 {x.z -> 42} \
 \text{\sffamily R1}
\end{phiquation*}
\end{document}
```

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

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

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

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

$$\begin{aligned} x(\pi) &\mapsto \llbracket\, \lambda \dashrightarrow f_1 \,\rrbracket, \\ x(a,b,c) &\mapsto \llbracket\, \alpha_0 \mapsto \varnothing, \varphi \mapsto \texttt{hello}(\xi) \,\rrbracket, \\ \Delta &= \texttt{43-09}. \end{aligned}$$

```latex
\documentclass{acmart}
\usepackage{eolang}
\thispagestyle{empty}
\begin{document}
\begin{phiquation*}
x(\pi) -> [[\lambda ..> f_1]], \\
x(a,b,c) -> [[ \alpha_0 -> ?, \
  @ -> |hello|($) ]], \\
\Delta = |43-09|.
\end{phiquation*}
\end{document}
```

4 Implementation

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

```latex
\RequirePackage{stmaryrd}
```

We need amsmath for `equation*` environment:

```latex
\RequirePackage{amsmath}
```

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

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

We need fancyvrb for `\VerbatimEnvironment` command:

```latex
\RequirePackage{fancyvrb}
```

We need iexec for executing Perl scripts:

```latex
\RequirePackage{iexec}
```

Then, we process package options:

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

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

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

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

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

`eolang-phi.pl` Then, we create a Perl script for `phiquation` processing:

```latex
\makeatletter
```

```
\begin{VerbatimOut}{\eolang@tmpdir/eolang-phi.pl}
$env = $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 ’% ’, $t, "\n";
}
print ’% ---’, "\n";
if ($env eq ’phiq’) {
  print ’$’;
} else {
  print ’\begin{’, $env, ’}\begin{split}’;
}
$tex =˜ s/ˆ\s+|\s+$//g;
if ($env ne ’phiq’) {
  $tex =˜ s/\s+\\\n\s*//g;
  $tex =˜ s/\\\\\n/\n\n/g;
}
$tex =˜ s/([\s,>\(])([0-9A-F][0-9A-F-]*)/\1|\2|/g;
$tex =˜ s/\?/\\varnothing{}/g;
$tex =˜ s/@/\\varphi{}/g;
$tex =˜ s/&/\\sigma{}/g;
$tex =˜ s/\ˆ/\\rho{}/g;
$tex =˜ s/\$/\\xi{}/g;
$tex =˜ s/-([a-z]+)>/\\mathrel{\\xrightarrow{\\text{\\sffamily\\scshape \1}}}/g;
$tex =˜ s/!->/\\mathrel{\\phiConst}/g;
$tex =˜ s/->/\\mathrel{\\mapsto}/g;
$tex =˜ s/:=/\\mathrel{\\vDash}/g;
$tex =˜ s/..>/\\mathrel{\\phiDotted}/g;
$tex =˜ s/\|([ˆ\|]+)\|/\\texttt{\1}{}/g;
$tex =˜ s/\[\[/\\llbracket\\mathrel{}/g;
$tex =˜ s/\]\]/\\mathrel{}\\rrbracket{}/g;
if ($env ne ’phiq’) {
  $tex =˜ s/\n\n/\\\\&/g;
  $tex =˜ s/\n/\\\\[-4pt]&/g;
  $tex =˜ s/([ˆ&\s])\s{2}([ˆ\s])/\1 \2/g;
  $tex =˜ s/\s{2}/ \\quad{}/g;
  my @leads = $tex =˜ /&[ˆ\s]+\s/g;
  my @eols = $tex =˜ /&/g;
  $tex = ’&’ . $tex;
  if (0+@leads == 0+@eols && 0+@eols > 0) {
    $tex =˜ s/&([ˆ\s]+)\s/\1&/g;
  }
}
print $tex;
```

```
if ($env eq ’phiq’) {
  print ’$’;
} else {
  print ’\end{split}\end{’, $env, ’}’;
}
print ’\endinput’, "\n";
\end{VerbatimOut}
\message{eolang: File with Perl script
  ’\eolang@tmpdir/eolang-phi.pl’ saved^^J}%
\iexec[trace,null]{perl -pi -e ’s/(\\\\[a-zA-Z])\\s+/\\1/g’
  "\eolang@tmpdir/eolang-phi.pl"}
\makeatother
```

`phiquation` Then, we define `phiquation` and `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"}%
  \iexec[trace,stdout=\eolang@tmpdir/\jobname/\hash-post.tex]{
    perl "\eolang@tmpdir/eolang-phi.pl"
    ’#1’
    "\eolang@tmpdir/\jobname/\hash.tex"}%
}
\newenvironment{phiquation*}%
{\VerbatimEnvironment\begin{VerbatimOut}
  {\eolang@tmpdir/\jobname/phiquation.tex}}
{\end{VerbatimOut}\eolang@process{equation*}}
\newenvironment{phiquation}%
{\VerbatimEnvironment\begin{VerbatimOut}
  {\eolang@tmpdir/\jobname/phiquation.tex}}
{\end{VerbatimOut}\eolang@process{equation}}
\makeatother
```

`\phiq` Then, we define `\phiq` command:

```
\makeatletter\newcommand\phiq[1]{
  \iexec[trace,quiet,stdout=\eolang@tmpdir/\jobname/phiq.tex]{
    /bin/echo ’\detokenize{#1}’}
  \def\hash{\eolang@mdfive
    {\eolang@tmpdir/\jobname/phiq.tex}}%
  \iexec[null]{cp "\eolang@tmpdir/\jobname/phiq.tex"
    "\eolang@tmpdir/\jobname/\hash.tex"}%
  \iexec[trace,stdout=\eolang@tmpdir/\jobname/\hash-post.tex]{
    perl \eolang@tmpdir/eolang-phi.pl ’phiq’
    "\eolang@tmpdir/\jobname/\hash.tex"}%
}\makeatother
```

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

```
\makeatletter
\begin{VerbatimOut}{\eolang@tmpdir/eolang-sodg.pl}
open(my $fh, ’<’, $ARGV[0]);
my $tex; { local $/; $tex = <$fh>; }
print ’% This file is auto-generated’, "\n";
print '% There are ', length($tex),
  ' chars in the input: ', $ARGV[0], "\n";
print '% ---', "\n";
if (index($tex, "\t") > 0) {
  print "TABS are prohibited!";
  exit 1;
}
$tex =~ s/^\s+|\s+$//g;
$tex =~ s/(\\[a-zA-Z]+)\s+/\1/g;
$tex =~ s/\n\s+/\n/g;
$tex =~ s/\|([^\|]+)\|/\\texttt{\1}/g;
my @cmds = split (/\n/g, $tex);
foreach my $t (@cmds) {
  print '% ', $t, "\n";
}
print '% ---', "\n";
print '\begin{phicture}', "\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 '') {
        my $d = $opts{'data'};
        if (index($d, '|') == -1) {
          $d = '\texttt{' . $d . '}';
        }
        $opts{'box'} = $d;
      }
    } elsif (exists $opts{'atom'}) {
      print ',phi-atom';
      if (not $opts{'atom'} eq '') {
        my $a = $opts{'atom'};
        if (index($a, '$') == -1) {
          $a = '$' . $a . '$';
        }
        $opts{'box'} = $a;
      }
    } 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{phicture}', "\n", '\endinput';
\end{VerbatimOut}
\message{eolang: File with Perl script
  '\eolang@tmpdir/eolang-sodg.pl' saved^^J}%
\iexec[trace,null]{perl -pi -e 's/(\\\\[a-zA-Z])\\s+/\\1/g'
  "\eolang@tmpdir/eolang-sodg.pl"}
\makeatother
```

`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}
```

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{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:

```
\makeatletter\newenvironment{sodg}%
{\VerbatimEnvironment\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"}%
  \iexec[trace,stdout=\eolang@tmpdir/\jobname/\hash-post.tex]{
    perl "\eolang@tmpdir/eolang-sodg.pl"
    "\eolang@tmpdir/\jobname/\hash.tex"}%
}\makeatother
```

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

```
\newcommand\eolang{%
  \ifdefined\anon%
```

```
    \anon[XYZ]{{\sffamily EO}}%
  \else%
    {\sffamily EO}%
  \fi%
}
```

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

```
\newcommand\phic{%
  \ifdefined\anon%
    \anon[$\alpha$-calculus]{$\varphi$-calculus}%
  \else%
    $\varphi$-calculus%
  \fi%
}
```

\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{%
  \ifdefined\anon%
    \anon[XML']{XMIR}%
  \else%
    XMIR%
  \fi%
}
```

\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}
```

\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.