`naive-ebnf`: LaTeX Package for EBNF in Plain Text[*]

Yegor Bugayenko
yegor256@gmail.com

2024/11/13, 0.0.18

NB! Large ENBF snippets may take too long to render!

1 Introduction

This package helps render an Extended Backus-Naur Form using plain text notation:

$$\begin{aligned}\langle\lambda\text{-Expr}\rangle \rightarrow\ & \langle\text{Var}\rangle \\ \mid\ & \text{“}\lambda\text{”}\ \langle\text{Var}\rangle\ \text{“.”}\ \langle\text{Expr}\rangle \\ \mid\ & \text{“(”}\ \langle\text{Expr}\rangle\ \langle\text{Expr}\rangle\ \text{“)”}\end{aligned}$$

```
\documentclass{minimal}
\usepackage{naive-ebnf}
\usepackage{mathtools}
\begin{document}
\begin{ebnf}
<$\lambda$-Expr> := <Var> \\
  || "$\lambda$" <Var> "." <Expr> \\
  || "\char‘\(" <Expr> <Expr> "\char‘\)"
\end{ebnf}
\end{document}
```

`ebnf` The `ebnf` environment *doesn’t* add any formatting to the paragraph, but only replaces the plain text symbols, such as “`:=`” and “`<Var>`” with proper LaTeX commands. The following syntax is understood inside the `ebnf` environment:

- `:=` separates the left-hand side from the right-hand side of the production rule;
- `<...>` denotes a non-terminal (variable);
- `"..."` denotes a terminal symbol;
- `’...’` denotes a special non-printable terminal symbol, like `’EOL’`;
- `(...|...)` denotes a series of options to choose from;
- `/.../` denotes a regular expression, like `/[a-z]+/`;
- `[...]` denotes an optional substitution;
- `{...}` denotes a zero or more times repetition;
- `{...}+` denotes one or more times repetition;

---

[*] The sources are in GitHub at yegor256/naive-ebnf

- `|||` denotes an indent at the beginning of the string.
- `||` denotes an indented vertical bar at the beginning of the string.

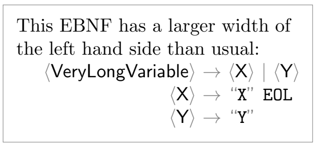
Attention: The usage of some symbols is prohibited inside terminals. Instead, the following substitutions are recommended:

- `$\lparen$` and `$\rparen$` instead of “`(`” and “`)`” (from the mathtools package);
- `$\langle$` and `$\rangle$` instead of “<” and “>”;
- `$\lbrace$` and `$\rbrace$` instead of “`{`” and “`}`” (also `mathtools`);
- `$\lbrack$` and `$\rbrack$` instead of “`[`” and “`]`” (also `mathtools`);
- `$\vert$` instead of “`|`”.

They would look even better, if the following notation is used:

- `\char‘\(` and `\char‘\)` instead of “`(`” and “`)`”;
- `\char‘\<` and `\char‘\>` instead of “<” and “>”;
- `\char‘\{` and `\char‘\}` instead of “`{`” and “`}`”;
- `\char‘\[` and `\char‘\]` instead of “`[`” and “`]`”.

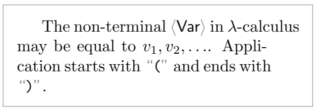
`width` There is an optional argument of `ebnf` environment, which sets the width of the left-hand side of each rule (the default width is `6em`):


This EBNF has a larger width of the left hand side than usual:
$\langle$VeryLongVariable$\rangle \to \langle$X$\rangle \mid \langle$Y$\rangle$
$\langle$X$\rangle \to$ “X” EOL
$\langle$Y$\rangle \to$ “Y”

```
This EBNF has a larger width of \\
the left hand side than usual: \par
\begin{ebnf}[1.5in]
<VeryLongVariable> := <X> | <Y> \\
<X> := "X" 'EOL' \\
<Y> := "Y" \\
\end{ebnf}
```

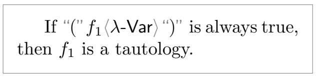
`\EbnfTerminal`
`\EbnfNonTerminal`
`\EbnfSpecial`

Inside the text, terminals, non-terminals, and special terminals may be formatted using three supplementary commands:


The non-terminal $\langle$Var$\rangle$ in λ-calculus may be equal to $v_1, v_2, \dots$. Application starts with “(” and ends with “)”.

```
The non-terminal \EbnfNonTerminal{Var}
in $\lambda$-calculus may be equal
to $v_1, v_2, \dots$. Application
starts with \EbnfTerminal{(} and ends
with \EbnfTerminal{)}.
```

It’s possible to use them in math-mode too, for example:


If “(” f_1 $\langle\lambda$-Var$\rangle$ “)” is always true, then f_1 is a tautology.

```
If $\EbnfTerminal{(} f_1
\EbnfNonTerminal{$\lambda$-Var}
\EbnfTerminal{)}$ is always true, then
$f_1$ is a tautology.
```

`\EbnfRegex` A regular expression is possible too:

⟨data⟩ → ⟨bool⟩ | ⟨integer⟩ | ⟨byte⟩
⟨bool⟩ → “TRUE” | “FALSE”
⟨integer⟩ → /(+|-)?[0-9]+/
⟨byte⟩ → /[0-9a-f]{2}/
⟨number⟩ → /[1-9]+/ /[0-9]+/

```
\begin{ebnf}
<data> := <bool> | <integer> | <byte> \\
<bool> := "TRUE" | "FALSE" \\
<integer> := /(+\char‘\|-)?[0-9]+/ \\
<byte> := /[0-9a-f]\char‘\{2\char‘\}/ \\
<number> := /[1-9]+/ /[0-9]+/
\end{ebnf}
```

Special symbols are interpreted correctly, if they stay inside quotes:

⟨X⟩ → EOL “'” “|”
⟨Y⟩ → “>” “<” “[” “]” “/” “/”
⟨Z⟩ → “LaTeX” “$”

```
\begin{ebnf}
<X> := ’EOL’ "’" "|" \\
<Y> := ">" "<" "[" "]" "/" "/" \\
<Z> := "\LaTeX" "\textdollar" \\
\end{ebnf}
```

Nested brackets work fine too:

⟨x⟩ → (“x” (“y” | (“z” | ⟨z⟩)))
⟨y⟩ → [[“x1”] {/[a-z]+/}]
⟨z⟩ → {{{⟨x⟩}⁺ ⟨y⟩} ⟨z⟩}⁺
⟨t⟩ → [⟨x⟩] [⟨y⟩]

```
\begin{ebnf}
% There is no meaning in this:
<x> := ( "x" ( "y" | ( "z" | <z> ) ) ) \\
<y> := [ [ "x1" ] { /[a-z]+/ } ] \\
<z> := { { { <x> }+ <y> } <z> }+ \\
<t> := [ <x> ] [ <y> ] \\
\end{ebnf}
```

The `|||` character allows indenting the text on a new line, allowing breaking long expressions:

⟨x⟩ → “beginning”
(⟨y⟩ | ⟨z⟩)
“ending”

```
\begin{ebnf}
<x> := "beginning"   \\
|||    ( <y> | <z> ) \\
|||    "ending"      \\
\end{ebnf}
\end{document}
```

2 Package Options

It’s possible to configure the behavior of the package with the help of a few package options:

`bw` By default, some colors are used in the rendered grammar. However, the `bw` package option disables any colors and makes sure the gammar is black-and-white:

```
\usepackage[bw]{naive-ebnf}
```

`trail` The `ebnf` environment is doing pre-processing of the TeX commands provided and then let LaTeX render them. It may be useful to see the output generated by the pre-processing. The `trail` option (with a file name) asks the package to save the content of the environment after the pre-processing into the file:

```
\usepackage[trail=log.tex]{naive-ebnf}
```

3 Implementation

First, we process package options:

```
\RequirePackage{pgfopts}
\pgfkeys{
  /ebnf/.cd,
  bw/.store in=\ebnf@bw,
  trail/.store in=\ebnf@trail,
  trail/.default=naive-ebnf.tmp.tex,
}
\ProcessPgfPackageOptions{/ebnf}
```

Then, we include a few packages, mostly to deal with LaTeX3 expressions:

```
\RequirePackage{expl3}
```

`\ebnf@color` Then, we include xcolor to colorize the output a bit:

```
\makeatletter\ifdefined\ebnf@bw\else
  \RequirePackage{xcolor}
\fi
\newcommand\ebnf@color[2]
  {\ifdefined\ebnf@bw#2\else\textcolor{#1}{#2}\fi}
\makeatother
```

`\EbnfTerminal` Then, we define a command to render a single terminal:

```
\makeatletter
\newcommand\EbnfTerminal[1]{{%
  \relax\ifmmode\else\ttfamily\fi%
  \ebnf@color{gray}{\relax\ifmmode\textsf{‘‘}\else{\sffamily‘‘}\fi}%
  #1%
  \ebnf@color{gray}{\relax\ifmmode\textsf{’’}\else{\sffamily’’}\fi}}
\makeatother
```

`\EbnfTerminal` Then, we define a command to render a single non-terminal:

```
\makeatletter
\newcommand\EbnfNonTerminal[1]{{%
  \ebnf@color{gray}{\relax\ifmmode\langle\else\(\langle\)\fi}%
  \relax\ifmmode\textsf{#1}\else{\sffamily#1}\fi%
  \ebnf@color{gray}{\relax\ifmmode\rangle\else\(\rangle\)\fi}}
\makeatother
```

`\EbnfSpecial` Then, we define a command to render a single non-terminal:

```
\makeatletter
\newcommand\EbnfSpecial[1]{{\relax\ifmmode\else\ttfamily\fi#1}}%
\makeatother
```

`\EbnfRegex` Then, we define a command to render a regular expression:

```
\makeatletter
\newcommand\EbnfRegex[1]{{\relax\ifmmode\else\ttfamily\fi/#1/}}%
\makeatother
```

Then, we define supplementary commands:

```
\makeatletter
\newcommand\ebnf@optional[1]
```

```
  {\ebnf@color{gray}{[}#1\ebnf@color{gray}{]}}
\newcommand\ebnf@repetition[2][]
  {\ebnf@color{gray}{\{}#2\ebnf@color{gray}{\}\(^{\scriptscriptstyle #1}\)}}
\newcommand\ebnf@grouping[1]
  {\ebnf@color{gray}{(}#1\ebnf@color{gray}{)}}
\ExplSyntaxOn
\newcommand\ebnf@terminal[1]{
  \tl_set:Nn \l_ebnf_tl {}
  \tl_set_rescan:Nnn \l_ebnf_tl {} { #1 }
  \EbnfTerminal{\l_ebnf_tl}
}
\newcommand\ebnf@special[1]{
  \tl_set:Nn \l_ebnf_tl {}
  \tl_set_rescan:Nnn \l_ebnf_tl {} { #1 }
  \EbnfSpecial{\l_ebnf_tl}
}
\newcommand\ebnf@nonterminal[1]{
  \tl_set:Nn \l_ebnf_tl {}
  \tl_set_rescan:Nnn \l_ebnf_tl {} { #1 }
  \EbnfNonTerminal{\l_ebnf_tl}
}
\newcommand\ebnf@regexp[1]{
  \tl_set:Nn \l_ebnf_tl {}
  \tl_set_rescan:Nnn \l_ebnf_tl {} { #1 }
  \EbnfRegex{\l_ebnf_tl}
}
\ExplSyntaxOff
\newcommand\ebnf@to
  {\ebnf@color{gray}{\(\to\)}}
\newcommand\ebnf@alternation
  {\ebnf@color{gray}{\(\vert\)}}
\makeatother
```

`ebnf` Then, we define the `ebnf` environment:

```
\ExplSyntaxOn
\cs_generate_variant:Nn \tl_replace_all:Nnn {Nx}
\makeatletter
\NewDocumentEnvironment{ebnf}{O{4em}+b}
  {\tl_set:Nn\ebnf_tmp{#2}}
  {%
  \regex_replace_all:nnN
    { ([^\s])/([^\s]) } {\1\\slash{}\2} \ebnf_tmp%
  \regex_replace_all:nnN
    { ([^\s])< } {\1\\textless{}} \ebnf_tmp%
  \regex_replace_all:nnN
    { >([^\s]) } {\\textgreater{}\1} \ebnf_tmp%
  \regex_replace_all:nnN
    { ([^\s])'([^\s]) } {\1\\textquotesingle{}\2} \ebnf_tmp%
  \regex_replace_all:nnN { \|\|\| }%
    {\c{makebox}[#1][r]{ }} \ebnf_tmp%
  \regex_replace_all:nnN
    { ([^\s])\|([^\s]) } {\1\\textbar{}\2} \ebnf_tmp%
  %
  \regex_replace_all:nnN
```

```
    { /(.+?)/ }%
    {\c{ebnf@regexp}{\1}} \ebnf_tmp%
  \cs_undefine:N\ebnf_curled%
  \cs_new:Npn\ebnf_curled{%
    \regex_replace_all:nnNT
    { \{\s(([^\s]*(\s[^\}\{]|\s(\}|\{)[^\s])?)*)\s\}([\+\*])? }%
    {\c{ebnf@repetition}[\5]{\1}} \ebnf_tmp \ebnf_curled}%
  \ebnf_curled%
  \cs_undefine:N\ebnf_brackets%
  \cs_new:Npn\ebnf_brackets{%
    \regex_replace_all:nnNT
    { \(\s(([^\s]*(\s[^\)\(]|\s(\)|\()[^\s])?)*)\s\) }%
    {\c{ebnf@grouping}{\1}} \ebnf_tmp \ebnf_brackets}%
  \ebnf_brackets%
  \cs_undefine:N\ebnf_squares%
  \cs_new:Npn\ebnf_squares{%
    \regex_replace_all:nnNT
    { \[\s(([^\s]*(\s[^\]\[]|\s(\]|\[)[^\s])?)*)\s\] }%
    {\c{ebnf@optional}{\1}} \ebnf_tmp \ebnf_squares}%
  \ebnf_squares%
  \regex_replace_all:nnN { (<[^>]+?>\s:=) }%
    {\c{makebox}[#1][r]{\1}} \ebnf_tmp%
  \regex_replace_all:nnN { <(.+?)> }%
    {\c{ebnf@nonterminal}{\1}} \ebnf_tmp%
  \regex_replace_all:nnN { "(.+?)" }%
    {\c{ebnf@terminal}{\1}} \ebnf_tmp%
  \regex_replace_all:nnN { ’(.+?)’ }%
    {\c{ebnf@special}{\1}} \ebnf_tmp%
  \regex_replace_all:nnN { \|(\|) }%
    {\c{makebox}[#1][r]{ \1 }} \ebnf_tmp%
  \regex_replace_all:nnN { \| }%
    {\c{ebnf@alternation}{}} \ebnf_tmp%
  \regex_replace_all:nnN { := }%
    {\c{ebnf@to}{}} \ebnf_tmp%
  \tl_put_left:Nn \ebnf_tmp {\noindent}
  \tl_put_right:Nn \ebnf_tmp {}
  \ifdefined\ebnf@trail%
    \newwrite\ebnf@write%
    \immediate\openout\ebnf@write\ebnf@trail\relax%
    \immediate\write\ebnf@write{\unexpanded\expandafter{\ebnf_tmp}}%
    \immediate\closeout\ebnf@write%
    \message{naive-ebnf:\space pre-processed\space TeX
      \space saved\space to\space "\ebnf@trail"^^J}%
  \fi%
  \ebnf_tmp}
\makeatother
\ExplSyntaxOff

\endinput
```

Change History

0.0.1
- General: First draft. 4

0.0.11
- `ebnf`: Many bugs fixed in the area of regular expression matching. 5

0.0.14
- `ebnf`: One-or-more repetition introduced with `{...}+` syntax. . . 5

0.0.15
- `ebnf`: The `iteration` removed, only `repetition` is left, with the second optional parameter. 5

0.0.2
- General: Proper parsing of grouping. . 4
  - Substitutions suggested for special symbols. 4
- `\EbnfTerminal`: New command `\EbnfNonTerminal` added, to enable rendering non-terminal symbols outside of the `ebnf` environment. 4
  - New command `\EbnfTerminal` added, to enable rendering terminal symbols outside of the `ebnf` environment. 4

0.0.3
- `\EbnfTerminal`: Quotes fixed in both text and math modes. 4

0.0.4
- `ebnf`: Any symbols are allowed inside `\EbnfNonTerminal` commands and inside the `ebnf` environment, where non-terminals are mentioned. 5

0.0.5
- General: New package option `trail` added, to enable saving of the generated TeX content to a file, for debugging purposes. 4

0.0.6
- `\EbnfSpecial`: New command `\EbnfSpecial` added, to enable rendering of special non-printable terminal symbols outside of the `ebnf` environment. 4

0.0.8
- `\EbnfRegex`: New command `\EbnfRegex` added, to enable rendering of regular expresions outside of the `ebnf` environment. . 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.