The plstx package

Jesse A. Tov
tov@ccs.neu.edu

This document corresponds to plstx v0.2, dated 2011/03/26.

Contents

1 Introduction

The purpose of this package is to provide a facility for typesetting grammars for programming language syntax, like this:

$$\begin{array}{rcll}
\alpha & \in & \mathit{TVar} & \textit{(type variables)} \\
x & \in & \mathit{Var} & \textit{(variables)} \\
\tau & ::= & \alpha \mid \tau_1 \to \tau_2 \mid \forall\alpha.\tau & \textit{(types)} \\
e & ::= & x \mid e_1\,e_2 \mid \lambda x\colon\tau.e \mid \Lambda\alpha.e \mid e[\tau] & \textit{(terms)}
\end{array}$$

Using the `plstx` environment, I coded that like this:

```
\begin{plstx}
  *(type variables): \alpha [\in] \mathit{TVar} \\
  *(variables):      x [\in] \mathit{Var} \\
  (types): \tau ::= \alpha | \tau_1 \to \tau_2 | \forall\alpha.\tau \\
  (terms): e    ::= x | e_1\,e_2 | \lambda x\colon\tau. e
                 | \Lambda\alpha.e | e[\tau] \\
\end{plstx}
```

The `plstx` environment allows redefining much of its behavior. For example, if we prefer $\longrightarrow$ to $::=$ in our grammars, we can change the "is one of" symbol. Perhaps we also want to change the formatting for the descriptions on the right.

```
\plstxset{
  is one of=\longrightarrow,
  label style=\textsf
}
```

Then we get:

$$\begin{array}{lcll}
\alpha & \in & \mathit{TVar} & \qquad\qquad\qquad\qquad \textsf{(type variables)} \\
x & \in & \mathit{Var} & \qquad\qquad\qquad\qquad \textsf{(variables)} \\
\tau & \longrightarrow & \alpha \mid \tau_1 \to \tau_2 \mid \forall\alpha.\tau & \qquad\qquad\qquad\qquad \textsf{(types)} \\
e & \longrightarrow & x \mid e_1\, e_2 \mid \lambda x{:}\,\tau.e \mid \Lambda\alpha.e \mid e[\tau] & \qquad\qquad\qquad\qquad \textsf{(terms)}
\end{array}$$

The environment also handles breaking lines when all the productions won't fit on one line, like this:

$$\begin{array}{lcll}
\alpha & \in & \mathit{TVar} & \textit{(type variables)} \\
x & \in & \mathit{Var} & \textit{(variables)} \\
\tau & ::= & \alpha \mid \tau_1 \to \tau_2 \mid \forall\alpha.\tau & \textit{(types)} \\
e & ::= & x \mid e_1\, e_2 \mid \lambda x{:}\,\tau.e & \textit{(terms)} \\
 & \mid & \Lambda\alpha.e \mid e[\tau] &
\end{array}$$

1.1 Requirements

The plstx package depends on three other packages. Two are a standard part of the LaTeX distribution: keyval and calc. The third, listproc, is non-standard, and may be obtained at `http://www.ccs.neu.edu/~tov/code/latex/`.

2 Command Reference

$\boxed{\texttt{\\plstxset \{}\langle \textit{plstx-options} \rangle\texttt{\}}}$

Takes a comma-separated list of keys and values, in the style of keyval:

$$\langle \textit{plstx-options} \rangle \; ::= \; \langle \textit{key} \rangle_1 \texttt{=} \langle \textit{value} \rangle_1 \texttt{,} \ldots \texttt{,} \langle \textit{key} \rangle_k \texttt{=} \langle \textit{value} \rangle_k [\texttt{,}]$$

The options available are described in §2.1.

```
\begin{plstx} [⟨plstx-options⟩]
  ⟨plstx-cmd⟩ ...
\end{plstx}
```

The `plstx` environment takes an optional argument, which is a list of options as keys and values, as described in §2.1. These are the same options that may be provided to `\plstxset`.

The available commands are:

⟨*plstx-cmd*⟩	::=	⟨*label-text*⟩`:` ⟨*nonterm*⟩ `::=` ⟨*rhs*⟩ `\\`
	|	`*` ⟨*label-text*⟩`:` ⟨*nonterm*⟩ `[`⟨*sep*⟩`]` ⟨*rhs*⟩ `\\`
	|	`\set {`⟨*plstx-options*⟩`}`
	|	`\intertext {`⟨*text*⟩`}`
	|	`[`⟨*dimen*⟩`]`

where

⟨*sep*⟩	::=	⟨*is-one-of*⟩
	|	⟨*is-one-of*⟩ `,` ⟨*continue*⟩
⟨*rhs*⟩	::=	⟨*production*⟩
	|	⟨*production*⟩ `|` ⟨*rhs*⟩

If a command starts with `*`, `\set`, `\intertext`, or `[`, then it is taken to be one of those four commands—otherwise, it is treated as the first case, which handles normal nonterminal item. We'll consider the available commands in order:

$\boxed{\langle \textit{label-text} \rangle\texttt{:}\ \langle \textit{nonterm} \rangle\ \texttt{::=}\ \langle \textit{rhs} \rangle\ \texttt{\\\\}}$

A normal nonterminal item consists of a label ⟨*label-text*⟩ (which is set on the right, in text mode by default); a non-terminal being defined ⟨*nonterm*⟩ (which is set on the left, in math mode by default); a separator (option `is one of`, default ::=, and written as `::=` in the command even if it has been configured to appear otherwise), and a right-hand side ⟨*rhs*⟩, which is a sequence of productions separated by `|`, each set in math mode by default. The nonterminal and label are set first, and then productions from the right-hand side are added one at a time until there's no more space remaining, at which point it may add continuation lines.

$\boxed{\texttt{*}\ \langle \textit{label-text} \rangle\texttt{:}\ \langle \textit{nonterm} \rangle\ \texttt{[}\langle \textit{sep} \rangle\texttt{]}\ \langle \textit{rhs} \rangle\ \texttt{\\\\}}$

A special nonterminal item starts with `*`, after which the syntax is the same as a normal nonterminal, with one exception. Rather than write `::=` for the "is one of" separator, it expects a separator for use in just that case to appear in square brackets. For example, to get $\alpha \in \mathit{TVar}$ in the example from §1, I wrote `\alpha [\in] \mathit{TVar}`. Optionally, the square brackets may contain a second item, after a comma, which indicates the separator to use for continuation lines if the right-hand side wraps. Writing a special `*` nonterminal item with separator `[::=,\vert]` is equivalent to writing a normal nonterminal.

`\set {⟨plstx-options⟩}`

This allows changing the options in the middle of a grammar, using the options described in §2.1. Changes made by `\set` last only until the end of the current `plstx` environment.

`\intertext {⟨text⟩}`

Escapes from the normal grammar typesetting to allow including arbitrary text between grammar items. (This is similar to amsmath's `\intertext` command.)

`[⟨dimen⟩]`

Inserts ⟨*dimen*⟩ vertical space.

Note: The grammar for ⟨plstx-cmd⟩ above was written like this:

```
\begin{plstx}[rhs style=,one per line]
  : \meta{plstx-cmd}
    ::= \meta{label-text}\texttt: \meta{nonterm} \defother{::=}
          \meta{rhs} \texttt{\bslash\bslash}
      | \defother{*} \meta{label-text}\texttt: \meta{nonterm}
          \texttt[\meta{sep}\texttt]
          \meta{rhs} \texttt{\bslash\bslash}
      | \defmacro{set} \marg{plstx-options}
      | \defmacro{intertext} \marg{text}
      | \defother{[}\meta{dimen}\texttt]
      | \oarg{dimen}
  \\
\intertext{where}
  : \meta{sep}
    ::= \meta{is-one-of}
      | \meta{is-one-of} \texttt, \meta{continue} \\
  : \meta{rhs}
    ::= \meta{production}
      | \meta{production} {\defother|} \meta{rhs} \\
\end{plstx}
```

2.1 Configuration Options

In this section, we document the configuration options that may be passed to `\plstxset`, `\set`, or environment `plstx`.

`align continue=⟨cs⟩`		*default:* '`\plstx@right`'
`continue center`	*(boolean)*	*default:* false
`continue left`	*(boolean)*	*default:* false
`continue right`	*(boolean)*	*default:* true

To configure the horizontal alignment of the continuation separator (see `continue`). The default is to right align it. It's possible to specify different alignment using one of the boolean options, or supply a command to format the continuation separator using `align continue`.

`align is one of=`⟨*cs*⟩		*default:* ‘`\plstx@center`’
`is one of center`	*(boolean)*	*default:* true
`is one of left`	*(boolean)*	*default:* false
`is one of right`	*(boolean)*	*default:* false

To configure the horizontal alignment of the “is one of” separator (see `is one of`). The default is to center it.

`align nonterm=`⟨*cs*⟩		*default:* ‘`\plstx@center`’
`nonterm center`	*(boolean)*	*default:* true
`nonterm left`	*(boolean)*	*default:* false
`nonterm right`	*(boolean)*	*default:* false

To configure the horizontal alignment of each nonterminal. The default is to center them.

`continue=`⟨*text*⟩ *default:* ‘`\vert`’

The “is one of” separator for continuation lines in normal grammar items. When the right-hand side spills onto additional lines, this is used in the separator column for each additional line. To change this for just one item, use the `*` command to get a special grammar item. The value of `continue` is set in math mode.

`continue center`	*(boolean)*	*see* `align continue`
`continue left`	*(boolean)*	*see* `align continue`
`continue right`	*(boolean)*	*see* `align continue`

`gutter=`⟨*dimen*⟩	*default:* ‘`4pt`’
`gutter left=`⟨*dimen*⟩	*default:* ‘`4pt`’
`gutter right=`⟨*dimen*⟩	*default:* ‘`4pt`’
`gutter left text=`⟨*text*⟩	*default:* ‘`\kern4pt`’
`gutter right text=`⟨*text*⟩	*default:* ‘`\kern4pt`’
`gutter text=`⟨*text*⟩	*default:* ‘`\kern4pt`’

These options are for specifying the *gutters*, which are the space to the left and right of the “is one of” separator. The `text` versions of the options set exactly what will be placed to the left or right (or both) of the separator, whereas the non-`text` versions allow supplying a length to be kerned. For example, each of these pairs is equivalent:

`gutter left=`⟨*dimen*⟩	≡	`gutter left text=\kern`⟨*dimen*⟩
`gutter right=`⟨*dimen*⟩	≡	`gutter right text=\kern`⟨*dimen*⟩
`gutter=`⟨*dimen*⟩	≡	`gutter left=`⟨*dimen*⟩`,gutter right=`⟨*dimen*⟩

`is one of=`⟨*text*⟩ *default:* ‘`::=`’

The separator for normal grammar items. To change this for just one item, use the `*` command to get a special grammar item. The value of `is one of` is set in math mode.

`is one of center` *(boolean)* *see* `align is one of`

`is one of left` *(boolean)* *see* `align is one of`

`is one of right` *(boolean)* *see* `align is one of`

`label skip=`⟨*dimen*⟩ *default:* ‘`1pc`’
`label skip text=`⟨*text*⟩ *default:* ‘`\kern1pc`’

This specifies the space to the left of the label, which separates the label from the right-hand side. Option `label skip text` takes the exact text to put to the left of (non-empty) labels, whereas `label skip` merely needs a length. The latter is defined in terms of the former: `label skip=`⟨*dimen*⟩ ≡ `label skip text=\kern`⟨*dimen*⟩.

`label style=`⟨*cs*⟩ *default:* ‘`\emph`’

Command used to style grammar labels. Providing this key with no value sets the option to empty.

`left margin=`⟨*dimen*⟩ *see* `margin`

`many per line` *(boolean)* *default:* true

Set as many right-hand side productions as will fit on each line before wrapping. This option does not take a value; the opposite option is `one per line`.

`margin=`⟨*dimen*⟩ *default:* ‘`0pt`’
`left margin=`⟨*dimen*⟩ *default:* ‘`0pt`’
`right margin=`⟨*dimen*⟩ *default:* ‘`0pt`’

Sets the margin on one or both sides of the grammar. This margin applies only to items (normal and special), not to `\intertext`. If no value is supplied, the margin is set to `1em`.

`nonterm center` *(boolean)* *see* `align nonterm`

`nonterm left` *(boolean)* *see* `align nonterm`

`nonterm right` *(boolean)* *see* `align nonterm`

`nonterm style=`⟨*cs*⟩ *default:* ‘`\ensuremath`’

Commands used to style nonterminals. By default, nonterminals are set in math mode using `\ensuremath`. Providing this key with no value sets the option to empty.

`one per line` *(boolean)* *default:* false

Set only one right-hand side production on each line, regardless of space. This option does not take a value; the opposite option is `many per line`.

`or=`⟨*text*⟩ *default:* '`\vert`'

Used to separate productions in a right-hand side. Set in math mode.

`or skip=`⟨*dimen*⟩ *default:* '`4pt`'
`or skip text=`⟨*text*⟩ *default:* '`\kern4pt`'

This specifies the space around the production separator (option `or`). Option `or skip text` takes the exact text to put on each side of the production separator, whereas `or skip` merely needs a length. The latter is defined in terms of the former: `or skip=`⟨*dimen*⟩ ≡ `or skip text=\kern`⟨*dimen*⟩.

`rhs style=`⟨*cs*⟩ *default:* '`\ensuremath`'

Commands used to style each right-hand side production. By default, productions are set in math mode using `\ensuremath`. Providing this key with no value sets the option to empty.

`right margin=`⟨*dimen*⟩ *see* `margin`

3 Implementation

We begin by requiring packages:

```
\RequirePackage{keyval}
\RequirePackage{calc}
\RequirePackage{listproc}
```

Set up the configuration options for keyval:

```
\define@key{plstx}{align continue}{\def\plstx@align@continue{#1}}
\define@key{plstx}{align is one of}{\def\plstx@align@isoneof{#1}}
\define@key{plstx}{align nonterm}{\def\plstx@align@nonterm{#1}}
\define@key{plstx}{continue center}[]{\def\plstx@align@continue{\plstx@center}}
\define@key{plstx}{continue left}[]{\def\plstx@align@continue{\plstx@left}}
\define@key{plstx}{continue right}[]{\def\plstx@align@continue{\plstx@right}}
\define@key{plstx}{continue}[]{\def\plstx@continue{#1}}
\define@key{plstx}{gutter}{%
  \def\plstx@gutter@left{\kern#1}%
  \def\plstx@gutter@right{\kern#1}}
\define@key{plstx}{gutter left text}{\def\plstx@gutter@left{#1}}
\define@key{plstx}{gutter left}{\def\plstx@gutter@left{\kern#1}}
\define@key{plstx}{gutter right text}{\def\plstx@gutter@right{#1}}
\define@key{plstx}{gutter right}{\def\plstx@gutter@right{\kern#1}}
\define@key{plstx}{gutter text}{%
  \def\plstx@gutter@left{#1}%
  \def\plstx@gutter@right{#1}}
\define@key{plstx}{is one of center}[]{\def\plstx@align@isoneof{\plstx@center}}
\define@key{plstx}{is one of left}[]{\def\plstx@align@isoneof{\plstx@left}}
\define@key{plstx}{is one of right}[]{\def\plstx@align@isoneof{\plstx@right}}
\define@key{plstx}{is one of}{\def\plstx@isoneof{#1}}
\define@key{plstx}{label skip text}{\def\plstx@labelskip{#1}}
\define@key{plstx}{label skip}{\def\plstx@labelskip{\kern#1}}
```

```
\define@key{plstx}{label style}[]{\def\plstx@label@style{#1}}
\define@key{plstx}{left margin}[1em]{\def\plstx@margin@left{\kern#1}}
\define@key{plstx}{many per line}[]{\let\plstx@one@per@line\@secondoftwo}
\define@key{plstx}{margin}[1em]{%
  \def\plstx@margin@left{\kern#1}%
  \def\plstx@margin@right{\kern#1}}
\define@key{plstx}{nonterm center}[]{\def\plstx@align@nonterm{\plstx@center}}
\define@key{plstx}{nonterm left}[]{\def\plstx@align@nonterm{\plstx@left}}
\define@key{plstx}{nonterm right}[]{\def\plstx@align@nonterm{\plstx@right}}
\define@key{plstx}{nonterm style}[]{\def\plstx@nonterm@style{#1}}
\define@key{plstx}{one per line}[]{\let\plstx@one@per@line\@firstoftwo}
\define@key{plstx}{or skip text}{\def\plstx@orskip{#1}}
\define@key{plstx}{or skip}{\def\plstx@orskip{\kern#1}}
\define@key{plstx}{or}{\def\plstx@or{#1}}
\define@key{plstx}{rhs style}[]{\def\plstx@rhs@style{#1}}
\define@key{plstx}{right margin}[1em]{\def\plstx@margin@right{\kern#1}}
```

`\plstx@set` To set configuration options, we delegate to `\setkeys` from the keyval package.

`\plstxset`

```
\newcommand*\plstx@set{\setkeys{plstx}}
\let\plstxset\plstx@set\relax
```

Set the initial options:

```
\plstx@set{
  continue      = \vert,
  continue right,
  gutter        = 4pt,
  is one of     = {::=},
  is one of center,
  label skip    = 1pc,
  label style   = \emph,
  many per line,
  margin        = 0pt,
  nonterm center,
  nonterm style = \ensuremath,
  or            = \vert,
  or skip       = 4pt,
  rhs style     = \ensuremath,
}
```

`\plstx@left` Helper commands for aligning text:

`\plstx@right`

`\plstx@center`

```
\def\plstx@left#1{#1\hfill}
\def\plstx@right#1{\hfill#1}
\def\plstx@center#1{\hfill#1\hfill}
```

`\plstx@parseRHS`

`\plstx@parseRHS@loop`

The right-hand side is provided by the user delimited by `|`. We need to break it into productions, carefully, in order to line break it as necessary. Command `\plstx@parseRHS` breaks `#1` into productions and stores them as a list in `#2` It does this by calling `\plstx@parseRHS@loop`, which uses TeX's argument pattern matching to find each `|`.

```
\newcommand\plstx@parseRHS[2]{%
  \let#1=\empty
  \plstx@parseRHS@loop#2|\plstx@parseRHS@stop\plstx@parseRHS@loop{#1}%
}
\def\plstx@parseRHS@loop#1|#2\plstx@parseRHS@loop#3{%
  \SnocTo{#1}{#3}%
  \ifx#2\plstx@parseRHS@stop
    \let\plstx@parseRHS@kont=\relax
  \else
     \def\plstx@parseRHS@kont{%
       \plstx@parseRHS@loop#2\plstx@parseRHS@loop{#3}%
     }%
  \fi
  \plstx@parseRHS@kont
}
```

`\plstx@additem` The `plstx` environment accumulates grammar items in a list, so that it can measure all of them before it chooses the widths of various parts. This macro adds an item to the accumulating list of items.

```
\newcommand\plstx@additem[1]{%
  \SnocTo{#1}{\plstx@items}%
}
```

`\plstx@dispatch` This macro is used inside the `plstx` environment to figure out which ⟨*plstx-command*⟩ comes next. It takes one argument, and then dispatches to the handler for the correct command. It has to deal with an additional case not mentioned in the user documentation: it detects the control sequences `\end` and `\endplstx` to detect when the environment is ending. If nothing matches, it dispatches to the normal item parser `\plstx@parseprod`.

```
\def\plstx@dispatch#1{%
  \ifx#1\end
    \let\plstx@dispatch@kont\end
  \else\ifx#1\endplstx
    \let\plstx@dispatch@kont\endplstx
  \else\ifx#1\intertext
    \let\plstx@dispatch@kont\plstx@intertext
  \else\ifx#1[%
    \let\plstx@dispatch@kont\plstx@vskip
  \else\ifx#1\set
    \let\plstx@dispatch@kont\plstx@set@later
  \else\ifx#1*%
    \let\plstx@dispatch@kont\plstx@other
  \else
    \def\plstx@dispatch@kont{\plstx@parseprod#1}%
  \fi\fi\fi\fi\fi\fi
  \plstx@dispatch@kont
}
```

`\plstx@parseprod` `::=` This is the command handler for normal productions. Productions are stored in the item list as

\plstx@production{⟨*label-text*⟩}{⟨*nonterm*⟩}{⟨*is-one-of*⟩}{⟨*continue*⟩}{⟨*rhs*⟩}

It then calls back to `\plstx@dispatch` to have it figure out the next command.

```
\def\plstx@parseprod#1:#2::=#3\\{%
  \plstx@additem{\plstx@production{#1}{#2}{\plstx@isoneof}{\plstx@continue}{#3}}%
  \plstx@dispatch%
}
```

`\plstx@other` `*` The command handler for special grammar items. Almost all the complexity is about figuring out whether the separator(s) in the square brackets are a single separator to use for both "is one of" and "continuation" separators, or two with a comma in between.

```
\def\plstx@other#1:#2[#3]#4\\{%
  \let\plstx@other@isoneof\plstx@isoneof
  \let\plstx@other@continue\plstx@continue
  \def\plstx@other@todo##1{%
    \def\plstx@other@isoneof{##1}%
    \def\plstx@other@continue{##1}%
    \def\plstx@other@todo####1{%
      \def\plstx@other@continue{####1}%
    }%
  }%
  \@for\plstx@each:=#3\do{%
    \expandafter\plstx@other@todo\expandafter{\plstx@each}%
  }%
  \def\plstx@other@addthis##1##2{%
    \plstx@additem{\plstx@production{#1}{#2}{##1}{##2}{#4}}%
  }%
  \expandafter\expandafter\expandafter\plstx@other@addthis
  \expandafter\expandafter
    \expandafter{\expandafter\plstx@other@isoneof\expandafter}%
    \expandafter{\plstx@other@continue}%
  \let\plstx@other@isoneof\@undefined
  \let\plstx@other@continue\@undefined
  \let\plstx@other@todo\@undefined
  \let\plstx@other@addthis\@undefined
  \plstx@dispatch
}
```

`\intertext` `\plstx@intertext` Intertext is added to the item list as

\plstx@intertext{⟨*text*⟩}

```
\def\plstx@intertext#1{%
  \plstx@additem{\plstx@intertext{#1}}%
  \plstx@dispatch%
}
```

`\plstx@vskip` `[⟨dimen⟩]` To add vertical space, we add

\plstx@later{\vskip⟨*dimen*⟩}

to the list of items.

```
\def\plstx@vskip#1]{\plstx@additem{\plstx@later{\vskip#1}}\plstx@dispatch}
```

`\set` `\plstx@set@later` For `\set`, we add `\plstx@set{`⟨*plstx-options*⟩`}` directly to the list of grammar items.

```
\def\plstx@set@later#1{\plstx@additem{\plstx@set{#1}}\plstx@dispatch}
```

`\plstx@box@a` `\plstx@box@b` `\plstx@box@c` We require three boxes: `box@a` is used for labels, `box@b` for the nonterminal and productions, and `box@c` as a temporary box as needed.

```
\newsavebox\plstx@box@a
\newsavebox\plstx@box@b
\newsavebox\plstx@box@c
```

`\plstx@maxnt` `\plstx@maxisoneof` `\plstx@availwd` We use two dimension registers for calculating the maximum width of the nonterminals and the maximum width of the “is one of” and “continue” separators. The third dimension register, `\plstx@availwd`, is used to keep track of remaining available width when line breaking the right-hand side.

```
\newlength\plstx@maxnt
\newlength\plstx@maxisoneof
\newlength\plstx@availwd
```

`plstx` The main `plstx` environment.

```
\newenvironment{plstx}[1][]
  {%
    \begingroup
```

Make sure that `|` is recognizable as the production separator:

```
    \catcode`\|=12\relax
    \plstx@set{#1}%
```

Initialize the list of items as empty. Then call `\plstx@dispatch` to read in the commands in the grammar.

```
    \let\plstx@items\empty
    \plstx@dispatch
  }
  {%
    \ifx\plstx@items\empty
      \PackageWarning{plstx}{grammar must have at least one production}%
    \else
```

For both passes through the list of items, we'll just evaluate the list, so we make `\listitem` a no-op.

```
      \def\plstx@listitem@noop##1{##1\let\listitem\plstx@listitem@noop}%
      \plstx@listitem@noop\relax%
```

We're going to compute the width of the widest nonterminal and widest “is one of.” We do this by defining `\plstx@production` to measure each nonterminal and “is one of.” The other grammar item callbacks are defined to do nothing for now.

```
      \setlength{\plstx@maxnt}{0pt}%
```

```
    \setlength{\plstx@maxisoneof}{0pt}%
    \def\plstx@production##1##2##3##4##5{%
      \setlength
        {\global\plstx@maxnt}
        {\maxof{\plstx@maxnt}{\widthof{\plstx@nonterm@style{##2}}}}%
      \setlength
        {\global\plstx@maxisoneof}
        {\maxof{\plstx@maxisoneof}
               {\maxof{\widthof{${##3}$}}
                      {\widthof{${##4}$}}}}%
    }%
    \def\plstx@intertext##1{}%
    \def\plstx@later##1{}%
    {\plstx@items}%
```

Now `\plstx@maxnt` is the widest nonterminal.

For the second pass, we actually output each item. We're going to wrap the whole thing in a `\trivlist`, so we'll precede each line with `\item`. We redefine the grammar item callbacks:

```
    \def\plstx@production##1##2##3##4##5{%
```

The initial available width is the `\linewidth`. We then add the label to `box@a`, and if the resulting box has non-zero width, we prepend `\plstx@labelskip` to it. Then, in either case, we postpend the right margin to it. We update the available width to account for the size of the label and any space around it.

```
      \setlength{\plstx@availwd}{\linewidth}%
      \sbox\plstx@box@a{\plstx@label@style{##1}}%
      \ifdim\wd\plstx@box@a>0pt
        \sbox\plstx@box@a{\plstx@labelskip\usebox\plstx@box@a}%
      \fi
      \sbox\plstx@box@a{\usebox\plstx@box@a\plstx@margin@right}%
      \addtolength{\plstx@availwd}{-\wd\plstx@box@a}%
```

Now we begin with the nonterminal. In `box@b`, we add the left margin, the nonterminal in a box of size `\plstx@maxnt` (formatted and aligned according to the options), the left gutter, the "is one of" separator, and finally the right gutter.

```
      \sbox\plstx@box@b{%
        \plstx@margin@left
        \makebox[\plstx@maxnt]
          {\plstx@align@nonterm{\plstx@nonterm@style{##2}}}%
        \plstx@gutter@left
        \makebox[\plstx@maxisoneof]{\plstx@align@isoneof{${##3}$}}%
        \plstx@gutter@right
      }%
```

Parse the right-hand side into a list of productions. We take the first production out of the list, postpend it to `box@b`, and update the available width.

```
      \plstx@parseRHS\plstx@rhsOut{##5}%
      \LopTo\plstx@rhsOut\plstx@rhsFirst
      \sbox\plstx@box@b
        {\usebox\plstx@box@b
```

```
     \plstx@rhs@style{\plstx@rhsFirst}}%
  \addtolength{\plstx@availwd}{-\wd\plstx@box@b}%
```

Now iterate over the remaining productions.

```
  \@forList\plstx@each:=\plstx@rhsOut\do{%
```

Place the next production in `box@c` along with the production separator. If option `one per line` is set, then we don't need to check, but otherwise, we check whether `box@c` will exceed the available space.

```
    \sbox\plstx@box@c
      {\plstx@orskip${\plstx@or}$\plstx@orskip
       \plstx@rhs@style{\plstx@each}}%
    \plstx@one@per@line
      {\iftrue}
      {\ifdim\wd\plstx@box@c>\plstx@availwd}%
```

In this case, either `box@c` won't fit or we're in one-per-line mode. So we stick `box@a` and `box@b` together and output them. Then, to start the next line, we reinitialize `box@a` with the right margin and `box@b` with the "continue" separator and the current production.

```
      \item\makebox[\linewidth]
        {\strut\usebox\plstx@box@b\hfill\usebox\plstx@box@a}
      \setlength{\plstx@availwd}{\linewidth}%
      \sbox\plstx@box@a{\plstx@margin@right}%
      \sbox\plstx@box@b{%
        \plstx@margin@left
        \makebox[\plstx@maxnt]{}%
        \plstx@gutter@left
        \makebox[\plstx@maxisoneof]{\plstx@align@continue{${##4}$}}%
        \plstx@gutter@right
        \plstx@rhs@style{\plstx@each}%
      }%
      \addtolength{\plstx@availwd}{-\wd\plstx@box@b}%
    \else
```

Otherwise, we add `box@c` to `box@b` and update the available width.

```
      \addtolength{\plstx@availwd}{-\wd\plstx@box@c}%
      \sbox\plstx@box@b{\usebox\plstx@box@b\usebox\plstx@box@c}%
    \fi
  }% end \do
```

When we've iterated through all the productions, we flush `box@b` if it isn't empty:

```
  \ifdim\wd\plstx@box@b>0pt
    \item\makebox[\linewidth]
      {\strut\usebox\plstx@box@b\hfill\usebox\plstx@box@a}
  \fi
}%
```

That's the end of the main grammar item callback.

For `\intertext`, we merely drop the text in a fresh `\item`. For items delayed with `\plstx@later`, we evaluate them as is.

```
      \def\plstx@intertext##1{%
        \item\strut\ignorespaces##1%
      }%
      \def\plstx@later##1{##1}%
```

Finally, we evaluate the list of grammar items in a `\trivlist`:

```
      \trivlist{\plstx@items}\endtrivlist
    \fi
    \endgroup
  }
```

Change History

v0.1
  General: Initial documented release . . . 1

v0.2
  General: Included listproc.sty 1

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.