Rail

Syntax Diagrams For LaTeX

L.W.J. Rooijakkers[*]
University of Nijmegen
The Netherlands
E-mail: `lwj@cs.kun.nl`

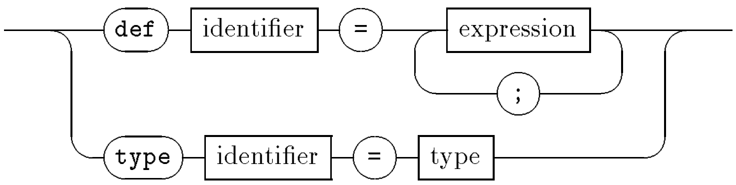
K. Barthelmann[†]
University of Mainz
Germany
E-mail: `barthel@informatik.uni-mainz.de`

December 13, 1996

1 Introduction

The Rail package allows you to include syntax diagrams (also known as *railroad diagrams*) in a LaTeX document. Such a diagram looks like this:

decl



The idea is that any sequence of terminals and nonterminals that can be produced by starting at the left and following the lines is a valid sentence of some language. As such, these diagrams are analogous to BNF with embedded regular expressions for each right-hand side. Actually, the input language looks like that, except that production rules need not be named and there is some extra annotation. This document describes version 1.1 of the Rail package.

2 Usage

To use the Rail package in your LaTeX document you need to write

[*]original version
[†]update

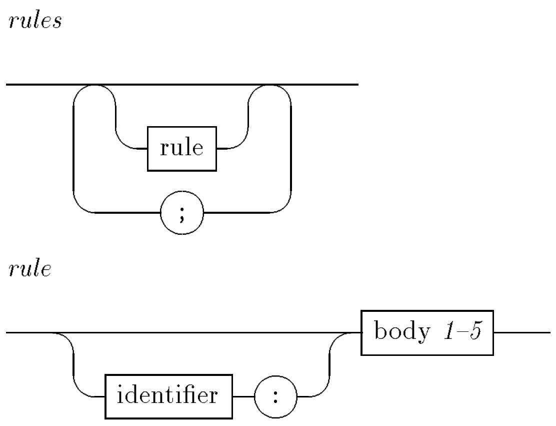


Figure 1: Input syntax for Rail

```
\usepackage{rail}
```

in the document's preamble. It is usually best to put the `rail` option last. Then you can display diagrams in your document body by using

`\begin{rail}`
rules
`\end{rail}`

where the syntax of *rules* is given in Figures 1 and 2, as a railroad diagram. There are also various options you can set in the preamble.

After running your *file*`.tex` through LaTeX, a file named *file*`.rai` will have been created, containing all the *rules* from your document. This file must then be processed with the `rail` program to produce *file*`.rao` containing LaTeX formatting instructions for each diagram This is done with the command

`rail` *file*

On the next LaTeX run, the diagrams will be picked up and integrated into the output. If the diagrams in *file*`.rao` are not up-to-date with your LaTeX file, the `rail` package will detect this and warn you with

`Package rail Warning: Railroad diagram {`*number*`} doesn't match on input line` *number*.

for each diagram, and again at the end of the document with

```
Package rail Warning: Railroad diagram(s) may have changed. Use 'rail' and rerun.
```

The *number* between braces can be used to find the diagram in *file*`.rai` or *file*`.rao` if needed.

3 Input language

As said before, diagrams are displayed by using

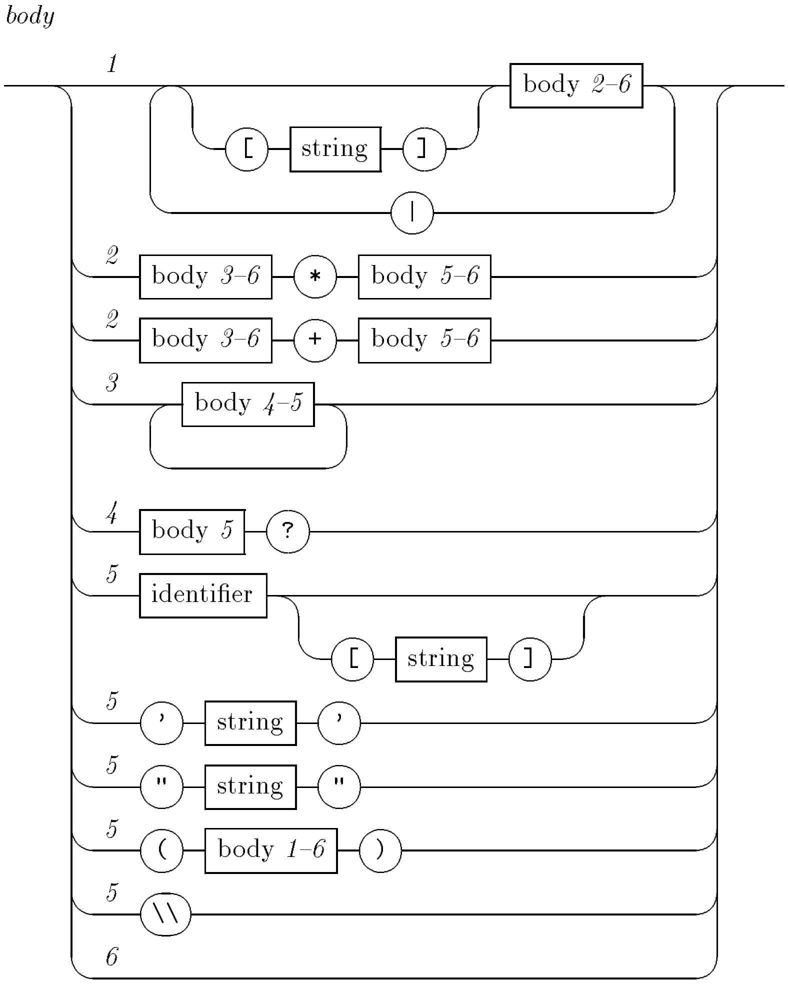
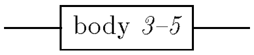


Figure 2: Input syntax for Rail, continued

```
\begin{rail}
rules
\end{rail}
```

Refer to Figures 1 and 2 for the syntax of *rules*. This syntax is rather strict. Some use of TEX macros within *rules* is possible, but no nested environments are allowed. However, defining a macro to save typing is possible.

As seen in the figure, *rules* consist of zero or more occurrences of a *rule*, separated by one or more semicolons. Each *rule* starts with an optional *identifier* giving the name of the rule, followed by the rule *body* (named rules can be indexed automatically with the `-i` option, see below) . This looks pretty much like BNF, but there are some other operators. Most of these are very similar to the usual operators of regular expressions, others are for formatting only. Parentheses `( )` are used for grouping. The italic numbers in the figure are the priority levels of each alternative. An occurrence like



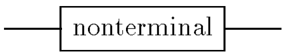
means that only alternatives with a priority in the range 3–5 are allowed at that point.

I will illustrate the meaning of the various operators with small examples.

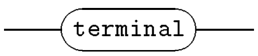
3.1 Atoms

The primitive atoms of the rules are identifiers and various forms of strings. Identifiers usually signify nonterminals, but they can also be used for terminal symbols (see `\railterm` below). Strings delimited by single quotes (`'`) or double quotes (`"`) signify terminals. Either kind of string may not contain tabs, newlines or it's closing quote. Note that these strings pass through TEX several times, thus it is unwise to use special characters like `{`, `}`, `$`, `%`, `&`, `\` in strings. If you need to use these as symbols, see `\railalias` below.

Any nonterminal is formatted in a square box, like



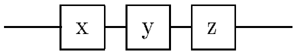
while any terminal text is formatted in a box with rounded corners like



Nonterminal identifiers can be annotated by following them with a string enclosed in `[` and `]`. This is described below under the `|` operator. The fonts used for typesetting terminals, nonterminals and annotations can be specified, see `\rail`...`font` below.

3.2 Concatenation

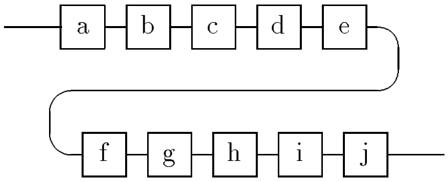
The most basic operation is concatenation, for which the operator is invisible. It works exactly like you think. Entering `x y z` produces



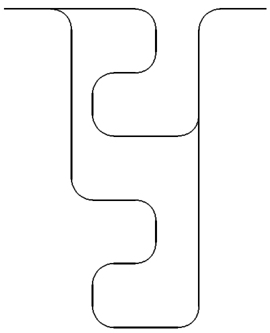
Related to concatenation is the empty body. It can occur only as operand of `|`, `*` or `+` or within parentheses. A safe way to write the empty body is `()`, which displays as

i.e., nothing. However, the invisible empty body is useful with the repetition operators, as will be seen below.

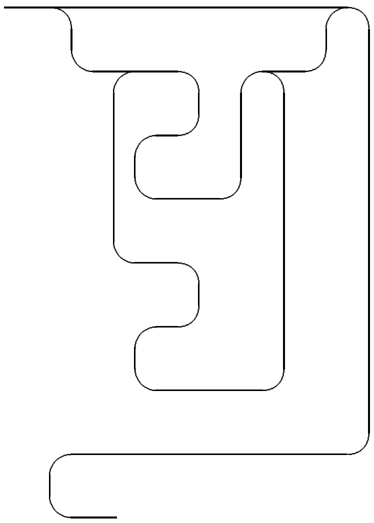
A feature related to concatenation is the ability to split long horizontal sequences with `\\`. As an example, `a b c d e \\ f g h i j` results in



It is possible to create horrible graphic constructions with this operator, for example `\\ | \\`:



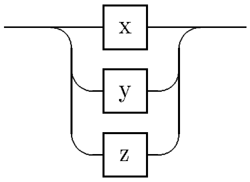
or `( \\ * \\ ) \\`:



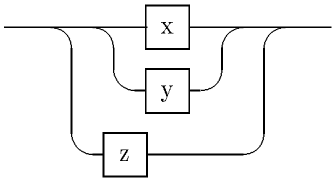
Nevertheless, every input should result in reasonable output.

3.3 Choice

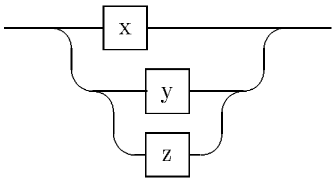
The next operation is choice, for which the operator is `|`. It has the lowest priority of all. Entering `x | y | z` produces



With this operator, parentheses are significant. The expressions `( x | y ) | z` and `x | ( y | z )` produce

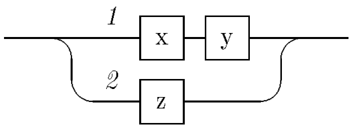


and

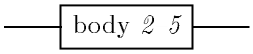


respectively, which have a different layout (but represent the same language).

Related to choice are priority annonations, of the form `[`*string*`]`. An example of their use can be seen in the railroad diagram for *body*. They should only be used with the top-level choice of a rule body, since otherwise their meaning is not clear. As an example, `[1] x y | [2] z` produces



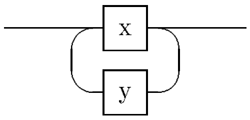
When used with identifiers as in `body[2--5]` they produce



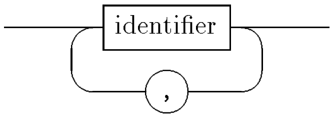
The *string* between `[` and `]` may not contain tabs, newlines or `]`. Remember that you should use `--` to produce a number dash (see *LaTeX: A Document Preparation System*, page 14).

3.4 Repetition

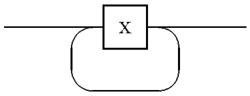
There are also the repetition operators `*` and `+`, which are similar to their regular-expression counterparts. For our purposes, `+` is the most basic one. The expression `x + y` means "one or more times `x`, separated by `y`," and is displayed as



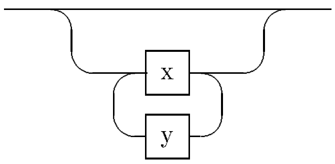
The utility of this construction is most obvious with things like



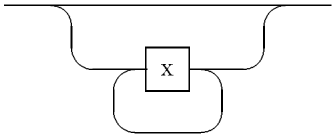
The second argument of `+` can be empty, resulting in the usual meaning of `x +` as "one or more times `x`," which is displayed as you would expect:



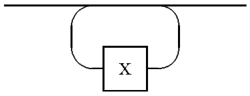
The Kleene star operator `*` is just a shorthand: `x * y` is identical in meaning to `() | x + y` and both display as



Analogously, `x *` is usually identical to `() | x +` and displays as



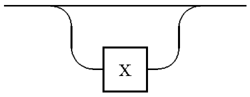
However, it is also possible to transform this to `() + x` which displays as



This can be made the default behaviour of Rail by using the `-a` option, either on the command line or with `\railoptions` (see below).

3.5 Optional things

Then there is the optionality operator `?`. The expression `x ?` is actually just a shorthand for `() | x` and displays like



3.6 An example

As an example, here is the input for the example diagram on page 1:

```
\begin{rail}

decl : 'def' identifier '=' ( expression + ';' )
     | 'type' identifier '=' type
     ;

\end{rail}
```

4 Formatting

4.1 Indexing

The Rail package can automatically index named rules. This is specified with the `-i` option of the `rail` command, which can be given on the command line or with `\railoptions`. If enabled, every rule of the form

identifier : *body*

will generate a LaTeX `\index` command for this identifier, normally in *italic* font (this can be changed with `\railindexfont`).

4.2 Terminal identifiers

It is possible to use identifiers as terminals, with optional user-specified formatting. To declare identifiers as terminals, use

`\railterm{`*identifier*`,`*identifier*`,`...`}`

To declare an identifier as a symbol with user-specified formatting, use

`\railalias{`*identifier*`}{`*text*`}`

Henceforth, *text* will be used to format the symbol named *identifier*. For example, the following commands can be used to use special TEX characters as terminals:

```
\railalias{lbrace}{\{}
\railalias{rbrace}{\}}
\railalias{dollar}{\$}
\railalias{percent}{\%}
\railalias{ampersand}{\&}
\railalias{backslash}{\char"5C}
\railterm{lbrace,rbrace,dollar,percent,ampersand,backslash}
```

4.3 Fonts

To format all text, the Rail package uses fonts that are all configurable with the following commands:

Text	Command	Default
rule names	`\railnamefont`	*italic font* (`\rmfamily\itshape`)
nonterminals	`\railnontermfont`	roman font (`\rmfamily\upshape`)
terminals	`\railtermfont`	`typewriter font` (`\ttfamily\upshape`)
annotations	`\railannotatefont`	*italic font* (`\rmfamily\itshape`)
index entries	`\railindexfont`	*italic font* (`\rmfamily\itshape`)

All of these are used like

`\rail`...`font{`*font*`}`

where *font* is a font control sequence like `\rmfamily` or a type style like `\bfseries` (it can actually be any sequence of formatting commands).

4.4 Style parameters

Every diagram is formatted as a LaTeX `picture` environment inside a `list` environment. The formatting parameters of the `list` environment can be set with

`\railparam{`*decls*`}`

where *decls* will be used as the *decls* argument of the `list` environment. For example,

```
\railparam{\addtolength{\itemsep}{1ex}}
```

increases the amount of vertical space between rules.

The `picture` line thickness is used for all lines in diagrams, so `\thinlines`, `\thicklines` and `\linethickness{`*len*`}` can be used to change it. However, `\unitlength` is not used (but see `\railunit` below).

Some of the formatting within the `picture` can be changed by modifying style parameters. If any of these are modified, the command `\railinit` should be executed before the next `rail` environment, otherwise some changes will not take effect. Otherwise, these parameters are subject to grouping.

`\railnamesep` The amount of vertical space between the rule name and the rule body.

`\railunit` This is the value of `\unitlength` used within diagrams. Usually set to `1sp` to provide maximal resolution.

`\railextra` The amount of extra line length added at the left and right ends of the diagram.

`\railboxheight` The height of the boxes enclosing terminals and nonterminals. This value is used as the size argument to `\oval`. For best results, only use multiples of `4pt`.

`\railboxskip` The amount of vertical space between the lines of a diagram.

`\railboxleft` The amount of extra line length added at the left of a box.

`\railboxright` Idem on the right.

`\railovalspace` The amount of extra space added to the size of the text to get the horizontal size of the oval box enclosing it.

`\railframespace` Idem for square boxes.

`\railtextleft` The amount of extra line length added at the left of an annotation.

`\railtextright` Idem at the right.

`\railtextup` The amount that annotation text is shifted up from the line it is attached to.

`\railjoinsize` The radius of the circle segments used to join and split lines. This value is used as the size argument to `\oval`. For best results, only use multiples of `4pt`.

5 The `rail` program

The full synopsis of the `rail` program is

`rail` [`+-acdit`] [*file*]

This will read *file*`.rai` and create *file*`.rao` if there are no errors. If there are any errors, *file*`.rao` will be removed. If no *file* argument is given, `rail` reads from standard input and writes to standard output.

5.1 Options

Option arguments start with a minus or a plus sign, followed by one or more option letters. If a minus sign is used, the options are set. With a plus sign, the options are reset. By default, no options are set. Options can also be set or reset from the LaTeX file (see below). This overrides the corresponding options setting from the command line. The effect of setting each option is described below.

`-a` An alternate layout is used for the `*` operator with an empty second argument. Instead of transforming `x * into () | x +` it is transformed into `() + x`.

`-c` The input is checked for undefined identifiers and unnamed rules. Statistics about those are printed to the standard output stream.

`-d` Turns on *yacc* debugging output. This only works if the program has been compiled with `YYDEBUG` defined (which is the default) and your *yacc* supports it.

`-i` Index entries are generated for all named rules, i.e., rules that are of the form

identifier : *body*

`-t` Print the parse tree of a rule body as comments in the `.rao` file.

Options can be set from a LaTeX file by use of

`\railoptions{`*options*`}`

where *options* is a set of option arguments just like those allowed on the command line. Options specified this way override those on the command line. The option settings take effect immediately.

Setting options this way is especially useful for the `-a`, `-c` and `-t` options. Messages about redefined identifiers are printed according to the setting of `-c` in effect at that point, but messages about undefined identifiers and unnamed rules are printed only if `-c` is still in effect at the end of the input file.

5.2 Manual page

There may also be an *nroff*/*troff* manual page available, which you can obtain by using `man rail`. This manual page is distributed with the Rail package as the file `rail.man`.

5.3 Bugs

Due to the use of a *yacc* parser, the error messages are not very helpful (essentially only '`syntax error`'), but this is difficult to correct.

6 Availability

As of version 1.1, the Rail package is available for Internet anonymous FTP or WWW from CTAN (Comprehensive TEX Archive Network) hosts in the directory `macros/latex/contrib/other/rail2e`.

Please report any bugs or complaints to the second author, K. Barthelmann. The first author, L.W.J. Rooijakkers, seems to be no longer reachable. Requests for features might be honored if I have the time (no chance `:-)`) or need the feature myself. Have fun!