Tau Prolog
Grammar specification

José A. Riaza Miguel Riaza

March 30, 2018

In this document, we describe the full Prolog grammar specification used by Tau Prolog to parse Prolog code.

Grammar

For simplicity, the terminal symbol `comma` in the grammar denotes an `atom` symbol (see Table 1) whose value is `','`. Furthermore, a terminal symbol with the form `op(specifier,priority)` denotes an atom with a given specifier[1] (`xf`, `yf`, `fx`, `fy`, `xfx`, `xfy` or `yfx`) and priority (between `0` and `1400`).

$$
\begin{array}{rcl}
\langle Expr_n \rangle & \rightarrow & \{\langle Expr_{n-1} \rangle.comma = \langle Expr_n \rangle.comma\}\ \texttt{op(fx,n)}\ \langle Expr_{n-1} \rangle \mid \\
& & \{\langle Expr_n \rangle_2.comma = \langle Expr_n \rangle.comma\}\ \texttt{op(fy,n)}\ \langle Expr_n \rangle_2 \mid \\
& & \{\langle Expr_{n-1} \rangle.comma = \langle Expr_n \rangle.comma\}\ \langle Expr_{n-1} \rangle\ \texttt{op(xf,n)} \mid \\
& & \{\langle Expr_n \rangle_2.comma = \langle Expr_n \rangle.comma\}\ \langle Expr_n \rangle_2\ \texttt{op(yf,n)} \mid \\
& & \{\langle Expr_{n-1} \rangle.comma = \langle Expr_{n-1} \rangle_2.comma = \langle Expr_n \rangle.comma\}\ \langle Expr_{n-1} \rangle\ \texttt{op(xfx,n)}\ \langle Expr_{n-1} \rangle_2 \mid \\
& & \{\langle Expr_n \rangle_2.comma = \langle Expr_{n-1} \rangle.comma = \langle Expr_n \rangle.comma\}\ \langle Expr_{n-1} \rangle\ \texttt{op(xfy,n)}\ \langle Expr_n \rangle_2 \mid \\
& & \{\langle Expr_n \rangle_2.comma = \langle Expr_{n-1} \rangle.comma = \langle Expr_n \rangle.comma\}\ \langle Expr_n \rangle_2\ \texttt{op(yfx,n)}\ \langle Expr_{n-1} \rangle \mid \\
& & \{\langle Expr_{n-1} \rangle.comma = \langle Expr_n \rangle.comma\}\ \langle Expr_{n-1} \rangle \\
\langle Expr_0 \rangle & \rightarrow & \texttt{number} \mid \texttt{variable} \mid \texttt{string} \mid \langle List \rangle \mid \langle Term \rangle \mid \\
& & \{\langle Expr_{1400} \rangle.comma = true\}\ \texttt{lparen}\ \langle Expr_{1400} \rangle\ \texttt{rparen} \mid \\
& & \{\langle Expr_{1400} \rangle.comma = true\}\ \texttt{lbrace}\ \langle Expr_{1400} \rangle\ \texttt{rbrace} \\
\langle Term \rangle & \rightarrow & \texttt{atom}\ \langle Term_2 \rangle \\
\langle Term_2 \rangle & \rightarrow & \{\langle Expr_{1400} \rangle.comma = false\}\ \texttt{lparen}\ \langle Expr_{1400} \rangle\ \langle Term_3 \rangle \mid \lambda \\
\langle Term_3 \rangle & \rightarrow & \{\langle Expr_{1400} \rangle.comma = false\}\ \texttt{comma}\ \langle Expr_{1400} \rangle\ \langle Term_3 \rangle \mid \texttt{rparen} \\
\langle List \rangle & \rightarrow & \texttt{lbracket}\ \langle List_2 \rangle \\
\langle List_2 \rangle & \rightarrow & \{\langle Expr_{1400} \rangle.comma = false\}\ \langle Expr_{1400} \rangle\ \langle List_3 \rangle \mid \texttt{rbracket} \\
\langle List_3 \rangle & \rightarrow & \{\langle Expr_{1400} \rangle.comma = false\}\ \texttt{comma}\ \langle Expr_{1400} \rangle\ \langle List_3 \rangle \mid \\
& & \{\langle Expr_{1400} \rangle.comma = false\}\ \texttt{bar}\ \langle Expr_{1400} \rangle\ \texttt{rbracket} \mid \\
& & \texttt{rbracket} \\
\langle Rule \rangle & \rightarrow & \{\langle Expr_{1400} \rangle.comma = true\}\ \langle Expr_{1400} \rangle\ \texttt{dot} \\
\langle Program \rangle & \rightarrow & \langle Rule \rangle\ \langle Program \rangle \mid \lambda
\end{array}
$$

The `comma` attribute of the $\langle Expr_n \rangle$ non-terminal symbol indicates whether a `comma` terminal symbol can be derived as an infix operator.

[1] The specifier indicates the type of operator (infix, prefix or sufix) and its associativity.

Terminal symbols

Table 1 shows all the terminal symbols of the grammar next to their regular expressions (with PCRE[2] syntax). Note that the `whitespace` symbol represents both white spaces and comments.

Table 1: Terminal symbols

Symbol	Regular expression
whitespace	/\s*(?:%.*|\/*(?:\n|\r|.)*?*\/|\s+)\s*/
variable	/[A-Z_][a-zA-Z0-9_]*/
dot	/\./
atom	/!|,|;|[a-z][0-9a-zA-Z_]*|[#\$\&*\+\-\.\/\:\<\=\>\?@\^\~\\]+| '(?:[^']*?(?:\\(?:x?\d+)?\\)*(?:'')*(?:\\')*)*'/
number	/0o[0-7]+|0x[0-9a-f]+|0b[01]+|0'(?:''|\\[abfnrtv\\'"`] |\\x?\d+\\\|.)|\d+(?:\.\d+(?:e[+-]?\d+)?)?/i
string	/"([^"]|""|\\")*"|`([^`]|``|\\`)*`/
lbrace	/\[/
rbrace	/\]/
lbracket	/\{/
rbracket	/\}/
lparen	/\(/
rparen	/\)/
bar	/\\|/
error	/./

Table 2 gives the initial operator table of Prolog, which can be modified using the `op/3` built-in predicate (more information about it in the url `http://tau-prolog.org/documentation/prolog/builtin/op/3`).

Table 2: Initial Prolog operators

Priority	Specifier	Operators
1200	xfx	`:-` `-->`
1200	fx	`:-` `?-`
1100	xfy	`;`
1050	xfy	`->`
1000	xfy	`,`
900	fy	`\+`
700	xfx	`=` `\=`
700	xfx	`==` `\==` `@<` `@=<` `@>` `@>=`
700	xfx	`=..`
700	xfx	`is` `=:=` `=\=` `<` `=<` `>` `>=`
500	yfx	`+` `-` `/\` `\/`
400	yfx	`*` `/` `//` `rem` `mod` `<<` `>>`
200	xfx	`**` `\\`
200	xfy	`^` `\\`
200	fy	`-` `\` `\\`

[2]Perl-compatible regular expressions.