BNF Syntax of Ruby

Based on `http://docs.huihoo.com/ruby/ruby-man-1.4/yacc.html` with some editing. I have found other websites attesting that Ruby version 1.4.6 is the latest with a reference manual in English, `http://docs.huihoo.com/ruby/ruby-man-1.4/index.html`, though Ruby exists up to v1.9 in Japanese.

ALL-CAPS are used for nonterminals, and all-lowercase for literal keywords. Literal `( ) [ ] { }` are quoted to distinguish them from BNF syntax.

```
PROGRAM    : COMPSTMT
T          : ";" | "\n"   //a newline can terminate a statement

COMPSTMT   : STMT {T EXPR} [T]

STMT       : CALL do ["|" [BLOCK_VAR] "|"] COMPSTMT end
           | undef FNAME
           | alias FNAME FNAME
           | STMT if EXPR
           | STMT while EXPR
           | STMT unless EXPR
           | STMT until EXPR
           | "BEGIN" "{" COMPSTMT "}"    //object initializer
           | "END" "{" COMPSTMT "}"      //object finalizer
           | LHS = COMMAND [do ["|" [BLOCK_VAR] "|"] COMPSTMT end]
           | EXPR

EXPR       : MLHS = MRHS
           | return CALL_ARGS
           | yield CALL_ARGS
           | EXPR and EXPR
           | EXPR or EXPR
           | not EXPR
           | COMMAND
           | ! COMMAND
           | ARG

CALL       : FUNCTION
           | COMMAND

COMMAND    : OPERATION CALL_ARGS
           | PRIMARY.OPERATION CALL_ARGS
           | PRIMARY :: OPERATION CALL_ARGS
           | super CALL_ARGS

FUNCTION   : OPERATION ["(" [CALL_ARGS] ")"]
           | PRIMARY.OPERATION "(" [CALL_ARGS] ")"
           | PRIMARY :: OPERATION "(" [CALL_ARGS] ")"
           | PRIMARY.OPERATION
           | PRIMARY :: OPERATION
           | super "(" [CALL_ARGS] ")"
           | super
```

```
ARG        : LHS = ARG
           | LHS OP_ASGN ARG
           | ARG .. ARG | ARG ... ARG
           | ARG + ARG | ARG - ARG | ARG * ARG | ARG / ARG
           | ARG % ARG | ARG ** ARG
           | + ARG | - ARG
           | ARG "|" ARG
           | ARG ˆ ARG | ARG & ARG
           | ARG <=> ARG
           | ARG > ARG | ARG >= ARG | ARG < ARG | ARG <= ARG
           | ARG == ARG | ARG === ARG | ARG != ARG
           | ARG =˜ ARG | ARG !˜ ARG
           | ! ARG | ˜ ARG
           | ARG << ARG | ARG >> ARG
           | ARG && ARG | ARG || ARG
           | defined? ARG
           | PRIMARY

PRIMARY: "(" COMPSTMT ")"
           | LITERAL
           | VARIABLE
           | PRIMARY :: IDENTIFIER
           | :: IDENTIFIER
           | PRIMARY "[" [ARGS] "]"
           | "[" [ARGS [,]] "]"
           | "{" [ARGS | ASSOCS [,]] "}"
           | return ["(" [CALL_ARGS] ")"]
           | yield ["(" [CALL_ARGS] ")"]
           | defined? "(" ARG ")"
           | FUNCTION
           | FUNCTION "{" ["|" [BLOCK_VAR] "|"] COMPSTMT "}"
           | if EXPR THEN
                COMPSTMT
             {elsif EXPR THEN
                COMPSTMT}
             [else
                COMPSTMT]
             end
           | unless EXPR THEN
                COMPSTMT
             [else
                COMPSTMT]
             end
           | while EXPR DO COMPSTMT end
           | until EXPR DO COMPSTMT end
           | case COMPSTMT
                when WHEN_ARGS THEN COMPSTMT
                {when WHEN_ARGS THEN COMPSTMT}
             [else
                COMPSTMT]
             end
```

```
            | for BLOCK_VAR in EXPR DO
                 COMPSTMT
              end
            | begin
                 COMPSTMT
              {rescue [ARGS] DO
                 COMPSTMT}
              [else
                 COMPSTMT]
              [ensure
                 COMPSTMT]
              end
            | class IDENTIFIER [< IDENTIFIER]
                 COMPSTMT
              end
            | module IDENTIFIER
                 COMPSTMT
              end
            | def FNAME ARGDECL
                 COMPSTMT
              end
            | def SINGLETON (. | ::) FNAME ARGDECL
                 COMPSTMT
              end

WHEN_ARGS   : ARGS [, * ARG]  |  * ARG

THEN        : T | then | T then  //"then" and "do" can go on next line
DO          : T | do   | T do

BLOCK_VAR   : LHS | MLHS

MLHS        : MLHS_ITEM , [MLHS_ITEM (, MLHS_ITEM)*] [* [LHS]]
            | * LHS

MLHS_ITEM   : LHS | "(" MLHS ")"

LHS         : VARIABLE
            | PRIMARY "[" [ARGS] "]"
            | PRIMARY.IDENTIFIER

MRHS        : ARGS [, * ARG]  |  * ARG

CALL_ARGS   : ARGS
            | ARGS [, ASSOCS] [, * ARG] [, & ARG]
            | ASSOCS [, * ARG] [, & ARG]
            | * ARG [, & ARG]  |  & ARG
            | COMMAND

ARGS        : ARG (, ARG)*
```

```
ARGDECL     : "(" ARGLIST ")"
            | ARGLIST T

ARGLIST     : IDENTIFIER(,IDENTIFIER)*[, *[IDENTIFIER]][,&IDENTIFIER]
            | *IDENTIFIER[, &IDENTIFIER]
            | [&IDENTIFIER]

SINGLETON   : VARIABLE
            | "(" EXPR ")"

ASSOCS      : ASSOC {, ASSOC}

ASSOC       : ARG => ARG

VARIABLE    : VARNAME | nil | self

LITERAL     : numeric | SYMBOL | STRING | STRING2 | HERE_DOC | REGEXP
```

The following are recognized by the lexical analyzer.

```
OP_ASGN     : += | -= | *= | /= | %= | **=
            | &= | |= | ˆ= | <<= | >>=
            | &&= | ||=

SYMBOL      : :FNAME | :VARNAME

FNAME       : IDENTIFIER | .. | "|" | ˆ | & | <=> | == | === | =˜
            | > | >= | < | <= | + | - | * | / | % | **
            | << | >> | ˜ | +@ | -@ | [] | []=

OPERATION   : IDENTIFIER [! | ?]

VARNAME     : GLOBAL | @IDENTIFIER | IDENTIFIER

GLOBAL      : $IDENTIFIER | $any_char | $-any_char

STRING      : " {any_char} "
            | ’ {any_char} ’
            | ‘ {any_char} ‘

STRING2     : %(Q|q|x)char {any_char} char

HERE_DOC    : <<(IDENTIFIER | STRING)
                {any_char}
                IDENTIFIER

REGEXP      : / {any_char} / [i|o|p]
            | %r char {any_char} char

IDENTIFIER : sequence in  /[a-zA-Z_]{a-zA-Z0-9_}/.
```