# Notação BNF Simplificada da linguagem ML

Este arquivo descreve a sintaxe BNF simplificada da linguagem ML (SML97). Os símbolos e construções escritos em letras maiúsculas são não-terminais e os escritos em letras minúsculas são literais. Os símbolos () [] são citados literalmente para distinguir da sintaxe.

```
PGM ::= EXP; | {DEC | MODULE | ;}
DEC ::= val [TVARS] PAT = EXP {and PAT = EXP}
      | val rec [TVARS] ID = fn MATCH {and ID = fn MATCH}
      | fun [TVARS] FBIND {and FBIND}
      | type TBINDS
      | local {DEC[;]} in {DEC[;]} end
      | open LID {LID}
      | infix[r] [INT] ID {ID}
      | nonfix ID {ID}

TVARS ::= TVAR | "("TVAR{,TVAR}")"
TBINDS ::= [TVARS] ID = TYPE {and [TVARS] ID = TYPE}
DTBIND ::= [TVARS] ID = ID [of TYPE] {"|" ID [of TYPE]}
FBIND ::= ID APAT {APAT} = EXP {"|" ID APAT {APAT} = EXP

TYPE ::= TVAR | RTYPE | BTYPE | LID
       | TYPE list
       | TYPE LID | "("TYPE{,TYPE}")"
       | TYPE * TYPE
       | TYPE -> TYPE
       | "("TYPE")"
RTYPE ::= "{"LABEL: TYPE{, LABEL: TYPE}"}"
BTYPE ::= int | char | string | bool
```

- **Expressões**

```
EXP ::= CONST | LID | PREFIXFN | PREFCON
      | #LABEL
      | "{"LABEL=EXP {, LABEL=EXP}"}"
      | "("EXP{, EXP}")"
      | "("EXP{; EXP}")"
      | "["EXP{, EXP}"]"
      | let {DEC[;]} in EXP{;EXP} end

      | EXP EXP
      | ! EXP
      | EXP INFIXFN EXP
      | EXP o EXP
      | EXP := EXP
      | EXP before EXP
      | EXP andalso EXP
      | EXP orelse EXP
      | if EXP then EXP else EXP
      | while EXP do EXP
      | case EXP of MATCH
      | fn MATCH
```

- **Símbolos literais**

```
LETTER ::= a | b | ... | z | A | B | ... | Z
SYMBOL ::= + - / * < > = ! @ # $ % ^ & ‘ ~ \ ? : "|"
DIGIT ::= 0 | 1 | ... | 9
HEXDIGIT ::= DIGIT | a | b | c | d | e | f
CHAR ::= #"<char>"
STRING ::= "anything in quotes"
INT ::= [~]DIGIT{DIGIT}
REAL ::= INT.DIGIT{DIGIT} | INT[.DIGIT{DIGIT}]"E"INT
WORD ::= 0wDIGIT{DIGIT} | 0wxHEXDIGIT{HEXDIGIT}

LID ::= {ID.}ID
ID ::= ALPHID | SYMBID
ALPHID ::= LETTER{LETTER | ’ | _}
SYMBID ::= SYMBOL{SYMBOL}
LABEL ::= ALPH_ID | SYMB_ID | DIGIT{DIGIT}
TVAR ::= ’ALPH_ID | ’’ALPH_ID (’’ for equality type)
CONST ::= INT | REAL| CHAR | STRING | WORD | () | MONOCON|MONOEXN
INFIXFN ::= * | / | div | mod
| + | - | ^
| @
| = | <> | < | > | <= | >=
| := | o
| ID
PREFIXFN::= op INFIXFN | ! | ~ | not | abs | LID
PREFCON ::= "SOME" | Fail : string->exn | LID
INFIXCON::= :: | ID
MONOCON ::= true | false | nil | "NONE" | "LESS" | "EQUAL" | "GREATER" |
LID
MONOEXN ::= Bind | Chr | Div | LID

concat : string list -> string;          chr : int -> char;
explode : string -> char list;           ord : char -> int;
implode : char list -> string;           str : char -> string;
substring : string*int*int -> string;    size : string -> int;
hd : ’a list -> ’a;                      tl : ’a list -> ’a list;
null : ’a list -> bool;                  rev : ’a list -> ’a list;
map : (’a -> ’b) -> ’a list -> ’b list;  length : ’a list -> int;
app : (’a -> unit) -> ’a list -> unit;
foldl,foldr : (’a * ’b -> ’b) -> ’b -> ’a list -> ’b list
```