

ml-ulex spec format¹

```
spec ::= ( declaration ';' )*
declaration ::= directive
              | rule
directive ::= '%charset' ( 'ASCII7' | 'ASCII8' | 'UTF8' )
              | '%defs' code
              | '%let' ID '=' re
              | '%name' ID
code ::= ' ( ' ... ' ) '
rule  ::= re '=>' code
re    ::= any nonreserved, nonwhitespace character or escape code
              | re '*'
              | re '?'
              | re '+'
              | '^' re
              | re '|' re
              | re '&' re
              | re '/' re
              | re '$'
              | STRING
              | '{' ID '}'
              | '[' '^'? ( char '-' char | char )+ ']'
              | re re
              | '(' re ')'
              | '.'
              | '_'
```

ml-antlr spec format

¹*ID* stands for `[A-Za-z][A-Za-z0-9_]*`. *STRING* is a double-quoted string, possibly including escape codes.

```

spec ::= ( declaration ';' )*
declaration ::= directive
              | nonterminal
directive ::= '%defs' code
              | '%import' STRING
              | '%keywords' symbol+
              | '%name' ID
              | '%start' ID
              | '%tokens' ':' tokdef ( '|' tokdef )*
code ::= ' ( ... ) '
tokdef ::= datacon ( ' ( STRING ) ' )?
datacon ::= ID
              | ID 'of' monotype
monotype ::= usual SML syntax
symbol ::= ID
              | STRING
nonterminal ::= ntdef
              | '%extend' ntdef
              | '%replace' ntdef
              | '%drop' ID+
ntdef ::= ID formals? ':' prodlist
formals ::= ' ( ID ( ',' ID )* ) '
prodlist ::= production ( '|' production )*
production ::= '%try'? named-item* ( '=>' code )? ( '%where' code )?
named-item ::= ( ID ':' )? item
item ::= prim-item '?'
              | prim-item '+'
              | prim-item '*'
prim-item ::= symbol args?
              | ' ( prodlist ) '
args ::= '@' code

```

An example

```
calc.lex

%name CalcLex;
%charset UTF8;

(* note: number and letter are predefined unicode
 * character classes.
 *)
%let int = [:number:]*;
%let id = [:letter:]([:letter:] | [:number:])*;

%defs (
    open CalcParse.Tok
);

let      => ( KW_let );
in       => ( KW_in );
{id}     => ( ID (yytext()) );
{int}    => ( NUM (valOf (Int.fromString (yytext())) ) );
"="      => ( EQ );
"+"      => ( PLUS );
"-"      => ( MINUS );
"*"      => ( TIMES );
"("      => ( LP );
")"      => ( RP );
[:whitespace:]
          => ( yyignore() );
.         => ( (* handle error *) );
```

```

calc.grm

%name CalcParse;
%tokens
  : KW_let  ("let")  | KW_in   ("in")
  | ID of string      | NUM of Int.int
  | EQ      ("=")     | PLUS   ("+")
  | TIMES   ("*")     | MINUS  ("-")
  | LP      "("       | RP     (")")
  ;

exp(env)
  : "let" ID "=" exp@(env)
    "in" exp@(AtomMap.insert(env, Atom.atom ID, exp1))
    => ( exp2 )
  | addExp@(env)
  ;

addExp(env)
  : multExp@(env) ("+" multExp@(env))*
    => ( List.foldl op+ multExp SR1 )
  ;

multExp(env)
  : prefixExp@(env) ("*" prefixExp@(env))*
    => ( List.foldl op* prefixExp SR1 )
  ;

prefixExp(env)
  : atomicExp@(env)
  | "-" prefixExp@(env)
    => ( ~prefixExp )
  ;

atomicExp(env)
  : ID
    => ( valOf(AtomMap.find (env, Atom.atom ID)) )
  | NUM
  | "(" exp@(env) ")"
  ;

```