ML Syntax

```
Phrase ::=
  [Exp | SimpleDecl | Module |
   "use" String | "monitor"] ";".

Module ::=
  "module" ModuleName
  ["includes" {ModuleName}1]
  "body" Decl "end".

SimpleExp ::=
  ["op"] (Ide | Con) |
  "[" {Exp / ";"} "]" |
  "(" {Exp / ";"}1 ")".

Exp ::=
  SimpleExp |
  Exp SimpleExp |
  Exp ":" Type |
  Exp Ide Exp |
  {Exp / ","}2 |
  "escape" Exp |
  "if" Exp "then" Exp "else" Exp |
  "while" Exp "do" Exp |
  "let" Decl "in" Exp "end" |
  Exp "where" Decl "end" |
  "case" Exp "of" Match |
  Exp "?" Exp |
  Exp "??" Exp Exp |
  Exp "?\" Ide "." Exp.
  "fun" Match.

SimpleDecl ::=
  "val" ValBind |
  "type" TypeBind |
  "abstype" TypeBind "with" Decl "end" |
  "export" ExportList "from" Decl "end" |
  "import" {ModuleName}1 |
  "infix" {Ide}1 | "nonfix" {Ide}1.

Decl ::=
  SimpleDecl |
  Decl ";" Decl |
  "(" Decl ")".

ExportList ::=
  {("abstype" | "type" | "val") {Ide / ","}1}1.

ValBind ::=
  Pat "=" Exp |
  ["op"] Ide {SimplePat}1 [":" Type] "=" Exp |
  {["op"] Ide SimplePat [":" Type] "=" Exp / "|"}2 |
  ValBind "and" ValBind |
  "rec" ValBind.

TypeBind ::=
  [Params] TypeIde "=" {Ide ["of" Type] / "|"}1 |
  TypeBind "and" TypeBind |
  "rec" TypeBind.

Params ::=
  TypeVar | "("{TypeVar / ","}1 ")".
```

Syntactic alternatives and infix operators are listed in order of decreasing precedence.

```
SimplePat ::=
  "_" |
  ["op"] Ide |
  Con |
  "[" {Pat / ";"} "]" |
  "(" Pat ")".

Pat ::=
  SimplePat |
  Con SimplePat |
  Pat ":" Type |
  SimplePat Con SimplePat |
  {Pat / ","}2 |

Match ::= {Pat "." Exp / "|"}1

Type ::=
  TypeVar |
  [TypeArgs] TypeIde |
  {Type / "*"}2 |
  Type "->" Type |
  "(" Type ")".

TypeArgs ::= Type | "(" {Type / ","}1 ")".

Letter ::= "a" | .. | "z" | "A" | .. | "Z" | "_".
Digit ::= "0" | .. | "9".
Symbol ::= !#%&$*+-/:<=>?@\^'|~.
Character ::= ht | .. | cf | " " | .. | "~".
Ide ::= Letter {Letter | Digit | "'"} | {Symbol}1.
Integer ::= {Digit}1
String ::= """ {Character} """.
Con ::= "(" ")" | Integer | String | Ide.
TypeIde ::= Ide.
TypeVar ::= "'" Ide.
ModuleName ::= Ide | String.
```

Keywords

? ?? ?\ = | _ : abstype and body case do else end export escape from fun import if in includes infix let local module nonfix of op rec then type use val where while with

Operations

{application}L {sub}L {* div mod}L {+ - ^}L {:: @}R {= <> > < >= <=}R {&}R {or o}R {user-infix}L {update :=}R true false not ~ size extract explode implode explodeascii implodeascii intofstring stringofint nil hd tl null length map rev fold revfold ref ! array arrayofilist lowerbound arraysize sub update arraytolist file save stream channel input output lookahead canInput terminal user-nonfix

Constructors

integers strings tuples () true false nil :: ref user-defined

Metasyntax

Strings between quotes ' " ' are terminals.
Identifiers are non-terminals.
Juxtaposition is syntactic concatenation.
' | ' is syntactic alternative.
'[..]' is zero or one times (i.e. optionally) ' .. '.
'{ .. }n' is n (default 0) or more times ' .. '.
'{ .. / -- }n' is n or more times ' .. ' separated by ' -- '.
Parentheses '(..)' are used for precedence.