1 Concrete Syntax

1.1 Notation Conventions

The following conventions are used for presenting the syntax of programs in *V*:

[⟨*pat*⟩]	optional pattern
{ ⟨*pat*⟩ }	zero or more occurrences of pattern
(⟨*pat*⟩)	grouping of patterns
⟨pat_1⟩ | ⟨pat_2⟩	choice
⟨*pat*⟩$_{\langle pat' \rangle}$	all patterns accepted by *pat*, except those also accepted by *pat′*
`text`	concrete characters (written in terminal font)

The syntax is described using a BNF grammar, with each production having the form:

⟨*sentence*⟩ ::= ⟨pat_1⟩ | ⟨pat_2⟩ | . . . | ⟨pat_n⟩

Whitespace is always explicitly expressed in productions with the ␣ character. It is used as a shorthand for the ⟨*whitespace*⟩ production. Literal characters will always be written in terminal font, so `|` and `[]` mean the literal characters, while | and [] are the choice and option pattern, respectively.

1.2 Basic Structure

⟨*program*⟩ ::= ⟨*whitespace*⟩ ⟨*expression*⟩ ⟨*eof*⟩

⟨*library*⟩ ::= ⟨*whitespace*⟩ { ⟨*declaration*⟩ } ⟨*eof*⟩

⟨*whitespace*⟩ ::= { ⟨*whitechars*⟩ | ⟨*comment*⟩ }

⟨*whitechars*⟩ ::= ⟨*whitechar*⟩ { ⟨*whitechar*⟩ }

⟨*whitechar*⟩ ::= ⟨*space*⟩ | ⟨*tab*⟩ | ⟨*newline*⟩

⟨*newline*⟩ ::= ⟨*return*⟩⟨*linefeed*⟩ | ⟨*return*⟩ | ⟨*linefeed*⟩

⟨*space*⟩ ::= space (' ')

⟨*tab*⟩ ::= horizontal tab ('\t')

⟨*return*⟩ ::= carriage return ('\r')

⟨*linefeed*⟩ ::= line feed ('\n')

⟨*comment*⟩ ::= `//` { ⟨*any*⟩$_{\langle newline \rangle}$ } ⟨*newline*⟩

⟨*any*⟩ ::= any ASCII character

1.3 Identifiers and Operators

⟨*identifier*⟩ ::= (⟨*idstart*⟩ { ⟨*idcontinue*⟩ })$_{\langle reservedid \rangle}$

⟨*idstart*⟩ ::= ⟨*small*⟩ | `_`

⟨*idcontinue*⟩ ::= ⟨*small*⟩ | ⟨*large*⟩ | ⟨*digit*⟩ | `'` | `_` | `?`

⟨*reservedid*⟩ ::= `let` | `true` | `false` | `if` | `then` | `else`
| `rec` | `nil` | `raise` | `when` | `match` | `with`
| `try` | `except` | `for` | `in` | `import` | `infix`
| `infixl` | `infixr` | `type` | `alias`

⟨*typeident*⟩ ::= (⟨*typeidentstart*⟩ { ⟨*idcontinue*⟩ })$_{\langle reservedtype \rangle}$

⟨*typeidentstart*⟩ ::= ⟨*large*⟩ | `_`

⟨*reservedtype*⟩ ::= `Int` | `Bool` | `Char`

⟨*small*⟩ ::= `a` | `b` | ... | `z`

⟨*large*⟩ ::= `A` | `B` | ... | `Z`

⟨*digit*⟩ ::= `0` | `1` | ... | `9`

⟨*operator*⟩ ::= ⟨*symbol*⟩ { ⟨*symbol*⟩ }

⟨*customop*⟩ ::= ⟨*operator*⟩$_{\langle reservedop \rangle}$

⟨*reservedop*⟩ ::= `+` | `-` | `*` | `/` | `<=` | `<` | `=`
| `!=` | `>=` | `>` | `::`

⟨*symbol*⟩ ::= `:` | `?` | `!` | `%` | `$` | `&` | `*` | `+` | `-`
| `.` | `/` | `<` | `=` | `>` | `@` | `^` | `|` | `~`

1.4 Terms

⟨*term*⟩ ::= ⟨*identifier*⟩
| `true` | `false` (booleans)
| ⟨*number*⟩
| `nil` (empty list)
| `raise` (exception)
| ⟨*char*⟩
| ⟨*string*⟩
| ⟨*parentheses*⟩
| ⟨*record*⟩
| `#` ⟨*identifier*⟩ (record access)
| ⟨*squarebrackets*⟩
| `if` ␣⟨*expression*⟩ `then` ␣⟨*expression*⟩ `else` ␣⟨*expression*⟩
| ⟨*match*⟩
| ⟨*lambda*⟩
| ⟨*reclambda*⟩
| ⟨*let*⟩

⟨*number*⟩ ::= ⟨*decimal*⟩ | ⟨*binary*⟩ | ⟨*octal*⟩ | ⟨*hexadecimal*⟩

⟨*decimal*⟩ ::= ⟨*digit*⟩ { ⟨*digit*⟩ }

⟨*binary*⟩ ::= `0` (`b` | `B`) ⟨*bindigit*⟩ { ⟨*bindigit*⟩ }

⟨*octal*⟩ ::= `0` (`o` | `O`) ⟨*octdigit*⟩ { ⟨*octdigit*⟩ }

⟨*hexadecimal*⟩ ::= `0` (`x` | `X`) ⟨*hexdigit*⟩ { ⟨*hexdigit*⟩ }

⟨*bindigit*⟩ ::= `0` | `1`

⟨*octdigit*⟩ ::= `0` | `1` | ... | `7`

⟨*hexdigit*⟩ ::= ⟨*digit*⟩ | `a` | ... | `f` | `A` | ... | `F`

⟨*char*⟩ ::= `'` (⟨*escape*⟩ | ⟨*any*⟩$_{\langle \text{'} | \backslash \rangle}$) `'`

⟨*string*⟩ ::= `"` { ⟨*escape*⟩ | ⟨*any*⟩$_{\langle \text{"} | \backslash \rangle}$ } `"`

⟨*escape*⟩ ::= `\` (`b` | `n` | `r` | `t` | `\` | `"` | `'`)

⟨*parentheses*⟩ ::= `(` ␣⟨*expression*⟩ `)`␣ (parenthesised expression)
| `(` ␣⟨*expression*⟩$_1$ `,` ␣... `,` ␣⟨*expression*⟩$_n$ `)` ␣ (tuple, $n \geq 2$)
| `(` ␣⟨*operator*⟩ ␣`)`␣ (prefix operator)

⟨*record*⟩ ::= `{` ␣⟨*recordcomp*⟩$_1$ `,` ␣... `,` ␣⟨*recordcomp*⟩$_n$ `}` ␣ ($n \geq 1$)

⟨*recordcomp*⟩ ::= ⟨*identifier*⟩ ␣`:` ␣⟨*expression*⟩

⟨*squarebrackets*⟩ ::= `[` ␣⟨*expression*⟩$_1$ `,` ␣`...` `,` ␣⟨*expression*⟩$_n$ `]` ␣ (list, $n \geq 0$)
| `[` ␣⟨*expression*⟩ `for` ␣⟨*pattern*⟩ `in` ⟨*expression*⟩ `]` ␣ (comprehension)
| `[` ␣⟨*expression*⟩ `[` `,` ␣⟨*expression*⟩ `]` `..` ␣⟨*expression*⟩ `]` ␣ (range)

⟨*match*⟩ ::= `match` ␣⟨*expression*⟩ `with` ␣`{` ⟨*matchcomp*⟩ `}`

⟨*matchcomp*⟩ ::= `|` ␣⟨*pattern*⟩ `[` `when` ␣⟨*expression*⟩ `]` `->` ␣⟨*expression*⟩

⟨*expression*⟩ ::= ⟨*operand*⟩ `{` ⟨*operator*⟩ ␣⟨*operand*⟩ `}`

⟨*operand*⟩ ::= ⟨*application*⟩
| `-` ␣⟨*application*⟩ (unary negation)

⟨*application*⟩ ::= ⟨*term*⟩ ␣`{` ⟨*term*⟩ ␣`}`

1.5 Functions and Declarations

⟨*lambda*⟩ ::= `\` ␣`{` ⟨*parameter*⟩ `}` `->` ␣⟨*expression*⟩

⟨*reclambda*⟩ ::= `rec` ␣⟨*identifier*⟩ ␣`{` ⟨*parameter*⟩ `}` `[` `:`␣⟨*type*⟩ `]` `->` ␣⟨*expression*⟩

⟨*let*⟩ ::= ⟨*declaration*⟩ `;` ␣⟨*expression*⟩

⟨*declaration*⟩ ::= `let` ␣⟨*pattern*⟩ `=` ␣⟨*expression*⟩
| `let` ␣`[` `rec` ␣`]` ⟨*funcname*⟩ `{` ⟨*parameter*⟩ `}` `[` `:` ␣⟨*type*⟩ `]` `=` ␣⟨*expression*⟩
| `import` ␣⟨*string*⟩ ␣
| `type` ␣`alias` ␣⟨*typeident*⟩ ␣`=` ␣⟨*type*⟩

⟨*funcname*⟩ ::= ⟨*identifier*⟩ ␣
| `[` ⟨*fixity*⟩ ␣⟨*digit*⟩ ␣`]` `(` ␣⟨*customop*⟩ ␣`)`␣ (operator declaration)

⟨*fixity*⟩ ::= `infixl` | `infixr` | `infix`

⟨*parameter*⟩ ::= ⟨*pattuple*⟩ ␣| `(` ␣⟨*pattern*⟩ `)` ␣| ⟨*patvalue*⟩ ␣

⟨*patvalue*⟩ ::= ⟨*identifier*⟩
| `_` (wildcard pattern)
| `true` | `false` (booleans)
| ⟨*number*⟩
| `nil` (empty list)
| ⟨*char*⟩
| ⟨*string*⟩
| ⟨*pattuple*⟩
| ⟨*patrecord*⟩
| `[` ␣⟨*pattern*⟩$_1$ `,` ␣`...` `,` ␣⟨*pattern*⟩$_n$ `]` ␣ (list, $n \geq 0$)

⟨*pattuple*⟩ ::= `(` ␣⟨*pattern*⟩$_1$ `,` ␣`...` `,` ␣⟨*pattern*⟩$_n$ `)` ␣ ($n \geq 2$)

⟨*patrecord*⟩ ::= `{` ␣⟨*patrecordcomp*⟩$_1$ `,` ␣`...` `,` ␣⟨*patrecordcomp*⟩$_n$ `}` ␣ ($n \geq 1$)
| `{` ␣⟨*patrecordcomp*⟩$_1$ `,` ␣`...` `,` ␣⟨*patrecordcomp*⟩$_n$ `,` `...` ␣`}` ␣
(partial record)

⟨*patrecordcomp*⟩ ::= ⟨*identifier*⟩ ␣`:` ␣⟨*pattern*⟩

⟨*pattern*⟩ ::= ⟨*patvalue*⟩ ␣`:` ␣⟨*type*⟩
| ⟨*patvalue*⟩ ␣`{` `::` ␣⟨*patvalue*⟩ ␣`}`

⟨*typevalue*⟩ ::= `Int`
| `Bool`
| `Char`
| ⟨*typeident*⟩
| `(` ␣⟨*type*⟩$_1$ `,` ␣`...` `,` ␣⟨*type*⟩$_n$ `)` ␣ (tuple, $n \geq 2$)
| `[` ␣⟨*type*⟩ ␣`]` ␣ (list)
| ⟨*typerecord*⟩

⟨*typerecord*⟩ ::= `{` ␣⟨*typerecordcomp*⟩$_1$ `,` ␣`...` `,` ␣⟨*typerecordcomp*⟩$_n$ `}` ␣ ($n \geq 1$)

⟨*typerecordcomp*⟩ ::= ⟨*identifier*⟩ ␣`:` ␣⟨*type*⟩

⟨*type*⟩ ::= ⟨*typevalue*⟩ ␣`{` `->` ⟨*typeValue*⟩ ␣`}`