The Complete Syntax of ♘

Alistair Lynn Sam Phippen

March 9, 2011

1 Lexemes

MODULE `([a-zA-Z][a-zA-Z0-9_-]+\.)*[a-zA-Z][a-zA-Z0-9_-]*`

SYMBOL `_*[a-z][a-zA-Z0-9_]*`

TYPENAME `_*[A-Z][a-zA-Z0-9_]*`

AT-SYMBOL `@[a-zA-Z0-9_]+`

DECIMAL-INTEGER `[0-9]+`

HEX-INTEGER `0[xX][0-9a-fA-F]+`

BINARY-INTEGER `0[bB][01]+`

OCTAL-INTEGER `0[oO][0-7]+`

FLOAT `[0-9]+\.[0-9]+(e-?[0-9]+)?`

STRING `"(\\"|[^"])*"`

CHARACTER `'\\.'|'[^\\]'`

2 Syntax

```
translation-unit ::= module-declaration? top-level-statement*

module-declaration ::= "module" MODULE ";"
```

```
top-level-statement ::= export-statement |
                        import-statement |
                        declaration      |
                        assignment

export-statement ::= "export" typename-list ";" |
                     "export" symbol-list ";"

typename-list ::= (TYPENAME ",")* TYPENAME

symbol-list ::= (SYMBOL ",")* SYMBOL

import-statement ::= "import" MODULE "." type ("as" TYPENAME)? ";"   |
                     "import" MODULE "." SYMBOL ("as" SYMBOL)? ";" |
                     "import" MODULE ".*" ";"

declaration ::= qualified-symbol "::" type ";"

qualified-symbol ::= SYMBOL type-qualification?

assignment ::= qualified-symbol "=" expression ";" |
               short-form-function

guarded-pattern-list ::= pattern-list guard?

short-form-function ::= qualified-symbol "(" guarded-pattern-list? ")"
                        function-body ";"

function-body ::= block |
                  ":=" expression

block ::= "{" statement* "}"

guard ::= "|" expression
```

```
type ::= TYPENAME type-qualification? |
         tuple-type                   |
         function-type                |
         enum-type                    |
         variant-type

variant-type ::= "variant" (variant-option ",")* variant-option

variant-option ::= TYPENAME tuple-type

function-type ::= type "->" type

enum-type ::= "enum" "{" enum-list "}"

enum-list ::= (AT-SYMBOL ",")* AT-SYMBOL

tuple-type ::= "(" (type ",")* type ")"

type-qualification ::= "<" type-parameter-list ">"

type-parameter-list ::= (type-parameter ",")* type-parameter

type-parameter ::= literal |
                   SYMBOL  |
                   type

statement ::= assignment               |
              declaration              |
              function-call-expression |
              block                    |
              while-block              |
              for-block                |
              do-while-block           |
              return-statement         |
              if-block                 |
              case-block               |
              ";"
```

```
return-statement ::= "return" expression ";"

while-block ::= "while" expression block

for-block ::= "for" pattern "in" expression block

do-while-block ::= "do" block "while" expression ";"

if-block ::= "if" expression block |
             "if" expression block "else" block

case-block ::= "case" expression "{" case-body* "}"

case-body ::= pattern-list block

pattern-list ::= (pattern ",")* pattern

pattern ::= SYMBOL                |
            "_"                   |
            pattern ":" pattern   |
            "[" pattern-list? "]" |
            "(" pattern-list? ")" |
            literal               |
            variant-match-pattern

variant-match-pattern ::= type |
                          type "(" pattern-list? ")"

expression ::= literal                        |
               function-call-expression       |
               list-expression                |
               tuple-expression               |
               hint-expression                |
               unary-expression               |
               binary-expression              |
               lambda                         |
               variant-constructor-expression |
               SYMBOL
```

```
function-call-expression ::= expression "(" expression-list? ")"

lambda ::= "\" "(" guarded-pattern-list? ")" function-body |
           "\" function-body

list-expression ::= "[" expression-list? "]"

expression-list ::= (expression ",")* expression

tuple-expression ::= "(" expression-list ")"

hint-expression ::= expression "::" type

unary-expression ::= unary-operator expression

binary-expression ::= expression binary-operator expression

literal ::= numeric-literal   |
            string-literal    |
            character-literal |
            symbol-literal

numeric-literal ::= DECIMAL-INTEGER |
                    HEX-INTEGER     |
                    BINARY-INTEGER  |
                    OCTAL-INTEGER   |
                    FLOAT

string-literal ::= STRING

character-literal ::= CHARACTER

symbol-literal ::= AT-SYMBOL

variant-constructor-expression ::= type |
                                   type tuple-expression
```

```
unary-operator ::= "!" |
                   "*" |
                   "-" |
                   "+"

binary-operator ::= "+"   |
                    "-"   |
                    "*"   |
                    "/"   |
                    "%"   |
                    "and" |
                    "or"  |
                    "=="  |
                    "!="  |
                    "<"   |
                    "<="  |
                    ">"   |
                    ">="  |
                    ":"   |
                    "<-"  |
                    "."   |
                    "^"   |
                    "!!"
```

3 Operator Precedences

Operator	**Associativity**
() {}	non-associative
. ^	right
calls and vcons	right
unary ! * - +	non-associative
!!	left
* / %	left
+ -	left
:	left
:: (type hint)	non-associative
`== != < <= > >=`	non-associative
and	left
or	left
`<-`	right