# The Complete Syntax of 🐴

Alistair Lynn  Sam Phippen

July 29, 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-stmts
                   | top-level-stmts
```

```
top-level-stmts ::= top-level-stmt top-level-stmts
                  | <epsilon>

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

top-level-stmt ::= export-stmt
                 | import-stmt
                 | variable-mutate-stmt
                 | type-definition-stmt

export-stmt ::= "export" exportee ";"

exportee ::= typename-list
           | symbol-list

typename-list ::= TYPENAME typename-list-rest

typename-list-rest ::= "," typename-list
                     | <epsilon>

symbol-list ::= SYMBOL symbol-list-rest

symbol-list-rest ::= "," symbol-list
                   | <epsilon>

import-stmt ::= "import" MODULE "." importee ";"

importee ::= TYPENAME optional-type-rename
           | SYMBOL optional-symbol-rename
           | "*";

optional-type-rename ::= "as" TYPENAME
                       | <epsilon>

optional-symbol-rename ::= "as" SYMBOL
                         | <epsilon>

qualified-symbol ::= SYMBOL optional-type-qualification
```

```
variable-mutation-stmt ::= qualified-symbol variable-mutation-suffix
                                            ";"

variable-mutation-suffix ::= "::" type
                           | "=" expr
                           | "(" optional-guarded-pattern-list ")"
                                function-body

optional-guarded-pattern-list ::= guarded-pattern-list
                                | <epsilon>

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

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

block ::= "{" stmts "}"

stmts ::= stmt stmts
        | <epsilon>

guard ::= "|" expr
        | <epsilon>

type-definition-stmt ::= possibly-qualified-named-type "=" type ";"

possibly-qualified-named-type ::= TYPENAME optional-type-qualification

type ::= function-type

parameter-type ::= parameter-function-type

function-type ::= base-type function-type-suffix

function-type-suffix ::= "->" base-type
                       | "," function-type
                       | <epsilon>
```

```
parameter-function-type ::= parameter-base-type parameter-function-type-suffix

parameter-function-type-suffix ::= "->" parameter-base-type
                                 | <epsilon>

base-type ::= possibly-qualified-named-type
            | tuple-type
            | enum-type
            | variant-type

parameter-base-type ::= possibly-qualified-named-type
                      | tuple-type
                      | enum-type

variant-type ::= "variant" variant-options

variant-options ::= variant-option variant-options-rest

variant-options-rest ::= "," variant-options
                       | <epsilon>

variant-option ::= TYPENAME variant-option-data

variant-option-data ::= tuple-type
                      | <epsilon>

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

enum-list ::= AT-SYMBOL enum-list-rest

enum-list-rest ::= "," enum-list
                 | <epsilon>

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

type-list ::= type type-list-rest
```

```
type-list-rest ::= "," type
                 | <epsilon>

optional-type-qualification ::= type-qualification
                              | <epsilon>

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

type-parameter-list ::= type-parameter type-parameter-list-rest

type-parameter-list-rest ::= "," type-parameter-list
                           | <epsilon>

type-parameter ::= literal
                 | SYMBOL
                 | parameter-type

stmt ::= variable-prefixed-stmt
       | block
       | while-block
       | for-block
       | do-while-block
       | return-stmt
       | if-block
       | case-block
       | ";"

variable-prefixed-stmt ::= SYMBOL variable-prefixed-stmt-suffix ";"

variable-prefixed-stmt-suffix ::= "::" type
                                | "=" expr
                                | call-family-suffix

return-stmt ::= "return" expr ";"

while-block ::= "while" expr block

for-block ::= "for" pattern "in" expr block
```

```
do-while-block ::= "do" block "while" expr ";"

if-block ::= "if" expr block else-suffix

else-suffix ::= "else" block
              | <epsilon>

case-block ::= "case" expr "{" case-innards "}"

case-innards ::= case-body case-innards
               | <epsilon>

case-body ::= pattern-list block

pattern-list ::= pattern pattern-list-rest

pattern-list-rest ::= "," pattern-list
                    | <epsilon>

optional-pattern-list ::= pattern-list
                        | <epsilon>

pattern ::= append-pattern

append-pattern ::= append-pattern ":" base-pattern
                 | base-pattern

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

variant-match-pattern ::= TYPENAME variant-match-pattern-data
```

```
variant-match-pattern-data ::= "(" optional-pattern-list ")"
                             | <epsilon>

expr ::= tuple-expr

tuple-expr ::= tuple-expr "," store-expr
             | store-expr

store-expr ::= or-expr store-suffix

store-suffix ::= "<-" store-expr
               | <epsilon>

or-expr ::= or-expr "or" and-expr
          | and-expr

and-expr ::= and-expr "and" rel-expr
           | rel-expr

rel-expr ::= hint-expr rel-suffix

rel-suffix ::= rel-operator hint-expr
             | <epsilon>

rel-operator ::= "=="
               | "!="
               | "<"
               | "<="
               | ">"
               | ">="

hint-expr ::= append-expr hint-suffix

hint-suffix ::= "::" type
              | <epsilon>

append-expr ::= append-expr ":" arithmetic-expr
              | arithmetic-expr
```

```
arithmetic-expr ::= arithmetic-expr arithmetic-operator
                                    multiplicative-expr
                  | multiplicative-expr

arithmetic-operator ::= "+"
                      | "-"

multiplicative-expr ::= multiplicative-expr multiplicative-operator
                                            array-index-expr
                      | array-index-expr

multiplicative-operator ::= "*"
                          | "/"
                          | "%"

array-index-expr ::= array-index-expr "!!" unary-expr
                   | unary-expr

unary-expr ::= call-family-expr
             | unary-operator unary-expr

call-family-expr ::= closure-operator-expr call-family-suffix
                   | variant-constructor call-family-suffix

call-family-suffix ::= call-arguments call-family-suffix
                     | <epsilon>

call-arguments ::= "(" optional-expr ")"

closure-operator-expr ::= base-expr closure-operator-suffix

closure-operator-suffix ::= closure-operator closure-operator-expr
                          | <epsilon>

closure-operator ::= "."
                   "^"
```

```
base-expr ::= lambda
            | literal
            | parenthesised-expr
            | list
            | SYMBOL

lambda ::= "\" lambda-arguments function-body

lambda-arguments ::= "(" optional-guarded-pattern-list ")"
                   | <epsilon>

list ::= "[" expr "]"

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 ::= TYPENAME variant-constructor-data

variant-constructor-data ::= parenthesised-expr
                           | <epsilon>

parenthesised-expr ::= "(" expr ")"

unary-operator ::= "!"
```

```
| "*"
| "-"
| "+"
```

# 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 |
| , | left |