

MiniML Grammar Spec

Brahima,Yukai,Zaid

2 fevrier, 2023

Contents

1	Change Log	2
2	Notes	2
2.1	Todo	2
3	Lexing Tokens	2
3.1	Separators	2
3.2	Mots-Clefs	2
3.3	Types	2
3.4	Operators	2
3.5	Valeurs_Atomiques	3
3.6	Identificateur	3
3.6.1	Constructeurs	3
4	Grammaire	4
4.1	Types	4
4.2	Expressions	4
4.3	Filtrage et Motifs	5
4.4	Definitions	5

1 Change Log

- 2 février, 2023 Première Version
- 2 février, 2023 Première Correction
 - Ajout de Unit
 - Ajout des patterns
 - Rename Value -> Litteral
 - Retrait Operators/Type de Base
 - Retrait Sucre Syntaxique pour le moment
- 7 février, 2023 Deuxième Correction
 - Simplification (des _LS)
 - Ajout des constructeurs infixes
 - Fix des Match Patterns
 - Fix Definition Globales
 - Reintroduction du Parsing Operators/Type de Base
 - Types Polymorphiques ?

2 Notes

2.1 Todo

- Crée du Sucre Syntaxique. # Plus Tard
- Parsing des types definis par l'utilisateur

3 Lexing Tokens

3.1 Separators

{ } [] () ; : , * -> | =

3.2 Mots-Clefs

let fun in match with type of

3.3 Types

int bool unit

3.4 Operators

+ - % / & | ~ :: && || *

3.5 Valeurs_Atomiques

```
nombre := ('-')?['0'-'9']*  
boolean := ("true"|"false")
```

3.6 Identificateur

```
alphanum := ['a'-'z' 'A'-'Z' '0'-'9' '_']*  
basic_ident := ['a'-'z' '_'] alphanum  
vartype := ['`t`'][0..9]*
```

3.6.1 Constructeurs

```
constructeur_ident := ['A'-'Z'] alphanum  
constructeur_infixes := [":": " ',']
```

4 Grammaire

```
Prog := | Def
      | Expr
      | Prog ;; Prog
```

4.1 Types

```
Type := | vartype
        | int
        | bool
        | unit
        | (Type)
        | Type * Type # Tuple_Type
        | ( Type list -> Type ) # Lambda_Type
```

4.2 Expressions

```
Litteral := | nombre
            | boolean
            | ( ) # Unit
            | [ ] # EmptyList
```

```
Variable := | basic_ident
            | basic_ident : Type
```

```
VarArgs := | Variable VarArgs
```

```
UnaryOperator := | ~
                 | -
```

```
BinaryOperator := | &
                  | &&
                  | ||
                  | +
                  | -
                  | /
                  | %
                  | *
```

Operator are translated to simple calls to STDLib

```
Expr := | ( Expr )
```

```

| Litteral
| Variable
| UnaryOperator Expr
| Expr BinaryOperator Expr
| constructeur_ident Expr # Built Expr
| constructeur_ident Built Expr
| Expr constructeur_infixes Expr
| Expr ; Expr # Sequence
| [ Expr ] # List
| let VarArgs = Expr in Expr # Binding
| fun VarArgs -> Expr # Lambda
| Expr Expr # Call
| match Expr with Match_Case

```

4.3 Filtrage et Motifs

```

Match_Case := | Pattern -> Expr
              | Pattern -> Expr '|' Match_Case

```

```

Pattern :=   | Litteral
              | basic_ident
              | _
              | constructeur_ident
              | constructeur_ident Pattern
              | Pattern constructeur_infixes Pattern
              | ( Pattern )

```

4.4 Definitions

```

Def      :=   | let VarArgs = Expr
              | type = ident NewConstructor_Case # Type Declaration

```

```

NewConstructor_Case := | constructeur_ident of Type
                      | constructor_ident
                      | NewConstructor_Case '|' NewConstructor_Case

```