

MiniML Grammar Spec

Brahima,Yukai,Zaid

2 fevrier, 2023

Contents

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

1 Change Log

- 2 fevrier, 2023 Première Version
- 2 fevrier, 2023 Première Correction
 - Ajout de Unit
 - Ajout des patterns
 - Rename Value -> Litteral
 - Retrait Operators/Type de Base
 - Retrait Sucre Syntaxique pour le moment
- 7 fevrier, 2023 Deuxième Correction
 - Simplification (des _LS)
 - Ajout des constructeurs infixes
 - Fix des Match Patterns
 - Fix Definition Globales
 - Reintroduction du Parsing Operators/Type de Base
- 11 fevrier, 2023 Post Reunion
 - Ajout et compréhension des vartypes
 - Ajout du keyword rec
 - Ajout des types paramettrer

2 Notes

2.1 Todo

- Crée du Sucre Syntaxique. # Plus Tard

3 Lexing Tokens

3.1 Separators

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

3.2 Mots-Clefs

`let fun in match with type of rec`

3.3 Types

`int bool unit`

3.4 Operators

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

3.5 Valeurs_Atomiques

`integer := ('-')?['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 Definitions

```
Def := | let Variable = Expr
      | type basic_ident = NewConstructor_Case # Type Def
      | type vartype basic_ident = NewConstructor_Case #ParametedType
```

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

4.2 Expressions

```
Litteral := | integer
            | boolean
            | ( ) # Unit
```

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

```
Expr := | ( Expr )
        | Litteral
        | Variable
        | UnaryOperator Expr
        | Expr BinaryOperator Expr
        | Expr ; Expr # Sequence
        | Expr Expr # Call
        | let Variable = Expr in Expr # Binding
        | fun Variable list -> Expr # Lambda
        | Expr constructeur_infixes Expr
        | constructeur_ident Expr # Built Expr
        | constructeur_ident # Avoid Nil ()
        | let Variable Variable list = Expr in Expr # func
        | let rec Variable Variable list = Expr in Expr # Recfunc
        | match Expr with Match_Case
```

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

```

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

```

4.3 Filtrage et Motifs

```

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

```

```

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

```

4.4 Types

```

Type := | (Type)
        | int
        | bool
        | unit
        | Type * Type # Tuple_Type
        | Type -> Type # Lambda_Type
        | vartype # 'a
        | Type List # Parametred Type (EXEMPLE : int list option)

```