MiniML Grammar Spec

Brahima,Yukai,Zaid

2 fevrier, 2023

Contents

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

2 Notes

2.1 Todo

- Crée du Sucre Syntaxique. # Plus Tard

3 Tokens

3.1 Symbols

3.2 Separators

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

3.3 Mots-Clefs

```
let rec fun in match with type of if then else
```

3.4 Valeurs_Atomiques

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

3.5 Identificateur

```
alphanum := ['a'-'z' 'A'-'Z' '0'-'9' '_']*
basic_ident := ['a'-'z' '_'] alphanum
constructeur_ident := ['A'-'Z'] alphanum
```

4 Grammaire

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

4.1 Variables

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

4.2 Types

```
Type    :=  | basic_ident
            | ( ) # Unit
            | [ ] # EmptyList_Type
            | ( Type_Ls ) # Tuple_Type
            | [ Type ] # List_Type
            | ( Types -> Type ) # Lambda_Type

Types   :=  | Type
            | Type Types
Type_Ls :=  | Type
            | Type , Type_Ls
```

4.3 Expressions

```
Litteral   :=  | nombre
               | boolean

Expr    :=  | ( Expr )
            | Litteral
            | Variable
            | constructeur_ident Expr # Custom Expr
            | ( ) # Unit
            | [ ] # EmptyList
            | ( Exprs_Ls ) # Tuple
            | [ Exprs_Ls ] # List
            | ( Exprs_Seq ) # Sequence
            | let Variable = Expr in Expr # Binding
            | fun Variables -> Expr # Lambda
```

```
            | if Expr then Expr else Expr # Condition
            | Expr (Exprs_Arg) # Call
            | match Expr with Match_Case


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

Patt :=     | Litteral
            | constructor_ident '(' Basic_Ident_LS ')'
            | ( )
            | ( Basic_Ident_LS )

Basic_Ident_LS :=   | basic_ident
                    | basic_ident , Basic_Ident_LS

Exprs_Arg := | Expr
             | Expr Exprs_Arg
Exprs_Ls  := | Expr
             | Expr , Exprs_Ls
Exprs_Seq := | Expr ; Exprs_Seq
```

4.4 Definitions

```
Def     :=   | let Variable = Expr
             | type = ident NewContructor_Case # Type Declaration

NewContructor_Case :=   | constructeur_ident of Type
                        | NewContructor '|' NewContructor_Case
```