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

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
polytype  := [’`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    :=  | polytype # Type Polymorph ?
            | int
            | bool
            | unit
            | (Type)
            | Type list
            | Type * Type # Tuple_Type
            | Type -> 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
            | 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
                | constructeur_ident
                | constructeur_ident Pattern
                | Pattern constructeur_infixes Pattern
                | ( Pattern )
```

4.4 Definitions

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

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