

# MiniML Spec

Fazazi Zeid

Luo Yukai

Dibassi Brahima

2023-05-06

# Table des matières

<b>1</b>	<b>Grammaire</b>	<b>3</b>
1.1	Elements Nommés . . . . .	3
1.2	Programmes . . . . .	3
1.3	Définitions . . . . .	3
1.4	Expressions . . . . .	4
1.5	Filtrage et Motifs . . . . .	4
1.6	Types . . . . .	5
<b>2</b>	<b>Semantique de traduction</b>	<b>5</b>
2.1	Notation . . . . .	5
2.2	Programmes . . . . .	5
2.3	Définitions . . . . .	6
	2.3.1 Définitions de Constructeurs . . . . .	6
2.4	Types . . . . .	6
2.5	Expressions . . . . .	7
2.6	Motifs et Filtrage . . . . .	8

# 1 Grammaire

Voici la grammaire BNF du langage MiniML. Les extensions non compatibles avec Ocaml mais compatible avec **Autobill** sont en **Orange**.

## 1.1 Elements Nommés

$$\begin{aligned}\langle \text{Id} \rangle &::= [ \text{'a' - z' 'A - Z' '0 - 9' \_'} ]^+ \\ \langle \text{ConstructeurId} \rangle &::= [ \text{'A - Z' } \langle \text{Id} \rangle \\ \langle \text{Vartype} \rangle &::= \text{'' [ 'a - z' [ '0 - 9' ]*}\end{aligned}$$

## 1.2 Programmes

$$\begin{aligned}\langle \text{Prog} \rangle &::= | \langle \text{Expr} \rangle \\ &| \langle \text{Def} \rangle \text{';;'} \langle \text{Prog} \rangle\end{aligned}$$

## 1.3 Definitions

$$\begin{aligned}\langle \text{Def} \rangle &::= | \text{'let' } \langle \text{Id} \rangle \text{'=' } \langle \text{Expr} \rangle \\ &| \text{'type' } \langle \text{Vartype} \rangle^* \langle \text{Id} \rangle \text{'=' } \langle \text{NewConstructors} \rangle \\ \langle \text{NewConstructors} \rangle &::= | \langle \text{ConstructeurId} \rangle \text{'of' } \langle \text{Type} \rangle \\ &| \langle \text{NewConstructors} \rangle \text{'/' } \langle \text{NewConstructors} \rangle \\ \textit{Debut Extension} &| \langle \text{ConstructeurId} \rangle \text{'of' } \langle \text{Type} \rangle \text{'with' } \langle \text{ParamAssigns} \rangle \\ \langle \text{ParamAssigns} \rangle &::= | \langle \text{VarType} \rangle \text{'=' } \langle \text{ParmExpr} \rangle \\ &| \langle \text{ParamAssigns} \rangle \text{' ,' } \langle \text{ParamAssigns} \rangle \\ \langle \text{ParmExpr} \rangle &::= | \text{'1' } \\ &| \langle \text{VarType} \rangle \\ &| \langle \text{ParmExpr} \rangle \text{'+' } \langle \text{ParmExpr} \rangle \\ \textit{Fin Extension} &| \langle \text{ParmExpr} \rangle \text{'*' } \langle \text{ParmExpr} \rangle\end{aligned}$$

## 1.4 Expressions

$\langle \text{Litteral} \rangle ::= \mid \begin{array}{l} [ \text{'0' - '9'} ] + \\ [ \text{'true' } \mid \text{'false'} ] \\ \text{'(' ' ')} \end{array}$

$\langle \text{Expr} \rangle ::= \mid \begin{array}{l} \langle \text{Litteral} \rangle \\ \langle \text{Id} \rangle \\ \langle \text{UnaryOperator} \rangle \langle \text{Expr} \rangle \\ \langle \text{Expr} \rangle \langle \text{BinaryOperator} \rangle \langle \text{Expr} \rangle \\ \langle \text{Expr} \rangle \langle \text{Expr} \rangle \\ \langle \text{Expr} \rangle \text{' ' } \langle \text{Expr} \rangle \\ \text{'let' } \langle \text{Id} \rangle \text{'=' } \langle \text{Expr} \rangle \text{'in' } \langle \text{Expr} \rangle \\ \text{'fun' } \langle \text{Id} \rangle \text{'->' } \langle \text{Expr} \rangle \\ \text{'fun' 'rec' } \langle \text{Id} \rangle \langle \text{Id} \rangle \text{'->' } \langle \text{Expr} \rangle \\ \langle \text{ConstructeurId} \rangle \langle \text{Expr} \rangle \\ \text{'match' } \langle \text{Expr} \rangle \text{'with' } \langle \text{MatchCase} \rangle \end{array}$

$\langle \text{UnaryOperator} \rangle ::= \text{'not'}$

$\langle \text{BinaryOperator} \rangle ::= [ \text{'and' } \mid \text{'or' } \mid \text{'+' } \mid \text{'-' } \mid \text{'/' } \mid \text{'\%' } \mid \text{'*' } \mid \text{'<' } \mid \text{'>' } \mid \text{'=' } ]$

## 1.5 Filtrage et Motifs

$\langle \text{MatchCase} \rangle ::= \mid \begin{array}{l} \langle \text{Pattern} \rangle \text{'->' } \langle \text{Expr} \rangle \\ \langle \text{MatchCase} \rangle \text{'/' } \langle \text{MatchCase} \rangle \end{array}$

$\langle \text{Pattern} \rangle ::= \mid \begin{array}{l} \langle \text{Litteral} \rangle \\ \langle \text{Id} \rangle \\ \langle \text{Pattern} \rangle \text{' ' } \langle \text{Pattern} \rangle \\ \langle \text{ConstructeurId} \rangle \langle \text{Pattern} \rangle \end{array}$

## 1.6 Types

$$\begin{array}{lcl}
 \langle \text{Type} \rangle & ::= & | \quad \langle \text{Vartype} \rangle \\
 & & | \quad \langle \text{Id} \rangle \\
 & & | \quad \langle \text{Type} \rangle \quad \langle \text{Type} \rangle \\
 & & | \quad \langle \text{Type} \rangle \quad \text{'->'} \quad \langle \text{Type} \rangle \\
 & & | \quad \langle \text{Type} \rangle \quad \text{'*'} \quad \langle \text{Type} \rangle
 \end{array}$$

## 2 Semantique de traduction

### 2.1 Notation

$$\llbracket \text{let } v = e_1 \text{ in } e_2 \rrbracket_{Expr} \rightarrow \text{let } v = \llbracket e_1 \rrbracket \text{ in } \llbracket e_2 \rrbracket$$

- $\llbracket \_ \rrbracket_{Expr} \rightarrow$  est la traduction d'un noeud expr du langage MiniML vers le langage LCBPV
  - À l'intérieur des  $\llbracket \_ \rrbracket$ , on trouve les elements propre au langage MiniML qui sont traduit vers le langage LCBPV.
  - À l'exterieur des  $\llbracket \_ \rrbracket$ , on trouve les elements propre au langage LCBPV.
- $X_n$  est le n-ième sous-noeud de l'arbre de syntaxe abstrait.  
Selon les cas  $X$  peut être :
  - $e$  pour les expressions
  - $p$  pour les motifs
  - $t$  pour les types
  - $d$  pour les définitions
    - \*  $dc$  pour les définitions de constructeurs
  - $v$  pour les elements nommées
  - $l$  pour les litteraux
- $X_1 \dots X_n$  est la liste des sous-noeuds de type  $X$  allant de 1 à  $n$

### 2.2 Programmes

Un programme MiniML est une suite de taille arbitraire de définitions suivie d'une expression.

$$\llbracket d_1 \dots d_n e \rrbracket_{Prog} \rightarrow \llbracket d_1 \dots d_n \rrbracket \text{ return } \llbracket e \rrbracket$$

## 2.3 Définitions

On définit l'opération de traduction  $\llbracket \_ \rrbracket_{Def}$  selon les cas de construction des définitions précisées par la règle de grammaire : **<Def>**

On distingue deux noeuds de types **<Def>** :

- Les définitions de **Variables Globales**,
- Les définitions de **Types**.

$$(VARDEF) \quad \llbracket let\ v = e \rrbracket_{Def} \rightarrow let\ v = \llbracket e \rrbracket$$

$$(TYPDEF) \quad \llbracket type\ v_1 \dots v_N = dc_1 \dots dc_N \rrbracket_{Def} \\ \rightarrow data\llbracket v_1 \rrbracket(\llbracket \dots v_N \rrbracket : +) = \llbracket dc_1 \dots dc_N \rrbracket$$

### 2.3.1 Définitions de Constructeurs

On définit l'opération de traduction  $\llbracket \_ \rrbracket_{DefConstructors}$  selon les cas de construction précisées par la règle de grammaire : **<NewConstructors>**

On distingue deux noeuds de types **<NewConstructors>** :

- Les définitions de **Constructeurs Classiques** qui sont compatibles avec Ocaml.
- Les définitions de **Constructeurs Equationnels** qui sont une extension spécifique pour **Autobill** incompatible avec Ocaml et dont la traduction n'est pas encore totalement définie.

$$(MLCONSTRUCTDEF) \quad \llbracket v\ of\ t \rrbracket_{DefConstructors} \rightarrow v\llbracket t \rrbracket$$

## 2.4 Types

On définit l'opération de traduction  $\llbracket \_ \rrbracket_{Type}$  selon les cas de construction des types précisées par la règle de grammaire : **<Type>**

On distingue 4 noeuds de types *Type*:

- Les **Variables de types**.
- Les **Applications de types**.
- Les **Lambda**.
- Les **Tuples**.

$$(TVAR) \quad \llbracket v \rrbracket_{Type} \rightarrow v$$

$$(TAPP) \quad \llbracket t_1 t_2 \rrbracket_{Type} \rightarrow \llbracket t_1 \rrbracket \llbracket t_2 \rrbracket$$

$$(TCLOS) \quad \llbracket t_1 \rightarrow t_2 \rrbracket_{Type} \rightarrow Exp(Fun\llbracket t_1 \rrbracket \rightarrow Thunk\llbracket t_2 \rrbracket)$$

$$(TTUPLE) \quad \llbracket t_1 * t_2 \rrbracket_{Type} \rightarrow \llbracket t_1 \rrbracket \llbracket t_2 \rrbracket$$

## 2.5 Expressions

On définit l'opération de traduction  $\llbracket \_ \rrbracket_{Expr}$  selon les cas de construction des expressions précisées par la règle de grammaire :  $\langle \text{Expr} \rangle$

On distingue 7 noeuds d'intérêts de types  $\langle \text{Expr} \rangle$  :

- Les **Tuples**.
- Les **Appels de fonctions**.
- Les **Fixation**.
- Les **Lambda**.
- Les **Fonctions Recursive**.
- Les **Constructions**.
- Les **Correspondance de motifs**.

**Note:** On ne prend pas en compte les **Litteraux** et les **Variables** dans ce descriptif car leurs traductions sont directes.

$$\begin{aligned}
(\text{TUPLE}) \quad & \llbracket e_1, e_2 \rrbracket_{Expr} \rightarrow \text{Tuple}(\llbracket e_1 \rrbracket \llbracket e_2 \rrbracket) \\
(\text{LAMBDA}) \quad & \llbracket \text{fun } v \rightarrow e \rrbracket_{Expr} \rightarrow \text{exp}(\text{get} \mid \text{call}(v) \rightarrow \text{thunk} \llbracket e \rrbracket) \\
(\text{FUN REC}) \quad & \llbracket \text{fun rec } v_1 v_2 \rightarrow e \rrbracket_{Expr} \rightarrow \text{exp}(\text{rec } v_1 \text{ is } \text{get} \mid \text{call}(v_2) \rightarrow \text{thunk} \llbracket e \rrbracket) \\
(\text{CALL}) \quad & \llbracket e_1 e_2 \rrbracket_{Expr} \rightarrow \{ \\
& \quad \text{open exp } v_1 = \llbracket e_1 \rrbracket \\
& \quad \text{force thunk}(v_2) = (v_1).\text{call} \llbracket e_2 \rrbracket \\
& \quad \text{return } v_2 \\
& \} \\
(\text{BIND}) \quad & \llbracket \text{let } v = e_1 \text{ in } e_2 \rrbracket_{Expr} \rightarrow \{ \\
& \quad \text{let } v = \llbracket e_1 \rrbracket \\
& \quad \text{return } \llbracket e_2 \rrbracket \\
& \} \\
(\text{CONSTRUCT}) \quad & \llbracket v e \rrbracket_{Expr} \rightarrow v(\llbracket e \rrbracket) \\
(\text{MATCH}) \quad & \llbracket \text{match } e \text{ with } p_1 \dots p_N \rrbracket_{Expr} \rightarrow \text{match } \llbracket e \rrbracket \text{ with } \llbracket p_1 \rrbracket \dots \llbracket p_N \rrbracket \text{ end}
\end{aligned}$$

## 2.6 Motifs et Filtrage

On définit l'opération de traduction  $\llbracket \_ \rrbracket_{Case}$  selon les cas de construction des expressions précisées par la règle de grammaire :  $\langle \text{MatchCase} \rangle$

On distingue 4 noeuds d'intérêt de types  $\langle \text{MatchCase} \rangle$  :

- Les patterns sur **Litteraux**.
- Les patterns sur **Variables**.
- Les patterns sur **Tuple**.
- Les patterns sur **Constructeurs**.

$$\begin{aligned} \text{(PATTERN LIT)} \quad & \llbracket l \rightarrow e \rrbracket_{MatchCase} \rightarrow \llbracket l \rrbracket \rightarrow \llbracket e \rrbracket \\ \text{(PATTERN VAR)} \quad & \llbracket v \rightarrow e \rrbracket_{MatchCase} \rightarrow v \rightarrow \llbracket e \rrbracket \\ \text{(PATTERN TUPLE)} \quad & \llbracket v_1, v_2 \rightarrow e \rrbracket_{MatchCase} \rightarrow Tuple(v_1 \ v_2) \rightarrow \llbracket e \rrbracket \\ \text{(PATTERN CONSTR)} \quad & \llbracket v_1 \ v_2 \rightarrow e \rrbracket_{MatchCase} \rightarrow v_1 \ v_2 \rightarrow \llbracket e \rrbracket \end{aligned}$$

On notera que la traduction des patterns ne prend pas en compte les patterns profonds, c'est à dire les patterns imbriqués dans d'autres patterns. En effet, les patterns profonds ne sont pas encore supportés par le langage cible.