MiniML Spec

Fazazi Zeid Luo Yukai Dibassi Brahima

2023-05-06

Table des matières

1 Grammaire

Voici la grammaire BNF du language MiniML.

1.1 Identificateurs

<Id> ::= ['a - z' 'A - Z' '0 - 9' '_']*
<ConstructeurId> ::= ['A - Z'] <Id>
<Vartype> ::= ''' ['a - z'] ['0 - 9']*

1.2 Programmes

<Prog> ::= | <Expr>
| <Def> ';;' <Prog>

1.3 Definitions

<Def> ::= | 'let' <Id> '=' <Expr>
| 'type' <Vartype>* <Id> '=' <NewContructors>

<NewContructors> ::= | <ConstructeurId> 'of' <Type>
| <ConstructeurId> 'of' <Type> 'with' <ParamAssigns>
| <NewContructors> '|' <NewContructors>

<ParamAssigns> ::= | <VarType> '=' <ParmExpr>
| <ParamAssigns> ',' <ParamAssigns>
<ParmExpr> ::= | '1'
| <VarType>
| <ParmExpr> '+' <ParmExpr>
| <ParmExpr> '*' <ParmExpr>

1.4 Expressions

<Litteral> ::= | ['0 - 9']+
| ['true' | 'false']
| '(' ')'

<Expr> ::= | <Litteral>
| <Id>
| <Operator>
| '(' <Expr> <Expr> ')'
| 'let' <Id> '=' <Expr> 'in' <Expr>
| 'fun' <Id> '->' <Expr>
| 'fun' 'rec' <Id> <Id> '->' <Expr>
| <ConstructeurId> <Expr>*
| 'match' <Expr> 'with' <MatchCase>

<Operator> ::= ['+' | '-' | '/' | '%' | '*' | '<' | '>' | '=']

1.5 Filtrage et Motifs

<MatchCase> ::= | <Pattern> '->' <Expr>
| <MatchCase> '|' <MatchCase>

<Pattern> ::= | <Litteral>
| <Id>
| <ConstructeurId> <Pattern>*

1.6 Types

<Type> ::= | <Vartype>
| <Id>
| <Type>+
| <Type> '->' <Type>

2 Semantique de traduction

2.1 Notation

$[\![Tuple(e_1 e_2)]\!]_{Expr} \rightarrow Tuple([\![e_1]\!][\![e_2]\!])$

- $[\![\_]\!]_{Expr} \rightarrow$ est la traduction d'un noeud expr du langage MiniML vers le langage LCBPV
- En Marron les elements propre au langage MiniML
- En Vert 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 identificateurs
  - i pour les entiers
  - b pour les booléens
- $X_1 \dots X_n$ 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. La sémantique de LCBPV ne permettant pas de définir des variables globales comme en MiniML il est nécessaire de transformer le programme en une seule expression.

On définit l'operation de traduction $[\![\_]\!]_{Prog}$ selon les cas de construction des définitions précisées par la règle de grammaire : **<Prog>**

$$[\![Prog(d_1 \dots d_n\, e)]\!]_{Prog} \rightarrow Prog([\![d_1 \dots d_n]\!]\, Do([\![e]\!]))$$

2.3 Définitions

On définit l'operation de traduction $[\![\_]\!]_{Def}$ 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**.

$$\text{(VARDEF)} \quad [\![VarDef(v\ e)]\!]_{Def} \rightarrow InsLet(v, [\![e]\!])$$

$$\begin{aligned}\text{(TYPDEF)} \quad & [\![TypeDef(v\ t_1 \ldots t_N\ c_1 \ldots c_N)]\!]_{Def} \\ & \rightarrow TypDef(v\ [\![t_1 \ldots t_N]\!]\ DefDatatype([\![c_1 \ldots c_N]\!]))\end{aligned}$$

2.4 Types

On définit l'operation de traduction $[\![\_]\!]_{Type}$ selon les cas de construction des types précisées par la règle de grammaire : <**Type**>

On distingue 3 noeuds de types $Type$:

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

$$\text{(TVAR)} \quad [\![TypeVar(v)]\!]_{Type} \rightarrow TypVar(v)$$

$$\text{(TAPP)} \quad [\![TypeConstructor(v\ t_1\ \ldots\ t_N)]\!]_{Type} \rightarrow TypApp(v\ [\![t_1 \ldots t_N]\!])$$

$$\begin{aligned}\text{(TCLOS)} \quad & [\![TypeLambda(t_1\ t_2)]\!]_{Type} \\ & \rightarrow TypClosure(\\ & \qquad Exp \\ & \qquad TypFun([\![t_1]\!]\ TypThunk([\![t_2]\!])) \\ &)\end{aligned}$$

2.5 Expressions

On définit l'operation de traduction $[\![\_]\!]_{Expr}$ selon les cas de construction des expressions précisées par la règle de grammaire : **<Expr>**

On distingue 8 noeuds de types *Expr*:

- Les **Litteraux**.
- Les **Variables**.
- Les **Appels de fonctions**.
- Les **Fixation**.
- Les **Lambda**.
- Les **Fonctions Recursive**.
- Les **Constructions**.
- Les **Correspondance de modèle**.

$$
\begin{aligned}
\text{(INT)}\quad & [\![Integer(i)]\!]_{Expr} \rightarrow ExprInteger(i) \\
\text{(BOOL)}\quad & [\![Boolean(b)]\!]_{Expr} \rightarrow ExprConstructor(b) \\
\text{(UNIT)}\quad & [\![Unit]\!]_{Expr} \rightarrow ExprConstructor(Unit)
\end{aligned}
$$

$$
\begin{aligned}
\text{(VAR)}\quad & [\![Variable(v)]\!]_{Expr} \rightarrow ExprVar(v) \\
\text{(LAMBDA)}\quad & [\![Lambda(v\,e)]\!]_{Expr} \\
& \rightarrow ExprClosure(\\
& \qquad Exp \\
& \qquad ExprGet(\ GetPatTag\ (Call\ v\ ExprThunk([\![e]\!]))\ ) \\
&)
\end{aligned}
$$

$$
\begin{aligned}
\text{(FUN REC)}\quad & [\![FonctionRec(v_1\,v_2\,e)]\!]_{Expr} \\
& \rightarrow ExprClosure(\\
& \qquad Exp \\
& \qquad ExprRec(v_1 \\
& \qquad\qquad ExprGet(\ GetPatTag\ (Call\ v_2\ ExprThunk([\![e]\!]))\ ) \\
& \qquad) \\
&)
\end{aligned}
$$

$$
\begin{aligned}
\text{(CALL)}\quad & [\![Call(e_1 e_2)]\!]_{Expr} \\
& \rightarrow ExprBlock(\\
& \qquad InsOpen(v_1\ Exp\ [\![e_1]\!]) \\
& \qquad InsForce\ (v_2\ ExprMethod(\ Call\ ExprVar(v_1)\ [\![e_2]\!]\ )\ ) \\
& \qquad ExprVar(v_2) \\
&)
\end{aligned}
$$

$$\begin{aligned}
\text{(BIND)}\quad & [\![\mathit{Binding}(v\, e_1 e_2)]\!]_{Expr} \\
& \to \mathit{ExprBlock}(\\
& \qquad\qquad \mathit{InsLet}(v\ [\![e_1]\!]) \\
& \qquad\qquad [\![e_2]\!] \\
& \quad)
\end{aligned}$$

$$\text{(CONSTRUCT)}\quad [\![\mathit{Construct}(v\, e_1 \ldots e_N)]\!]_{Expr} \to \mathit{ExprConstructor}(\mathit{ConsNamed}(v)\ [\![e_1 \ldots e_N]\!])$$

$$\text{(MATCH)}\quad [\![\mathit{Match}(e\, p_1\ \ldots\ p_N)]\!]_{Expr} \to \mathit{ExprMatch}([\![e]\!]\ [\![p_1 \ldots p_N]\!])$$

2.6 Motifs et Filtrage

$$\mathit{Case}(p, e) \to [\![\mathit{Case}(p, e)]\!]_{Case} \to \alpha$$

On définit la relation *Case* selon les cas de construction des motif de correspondance. Les cas de construction des motif de correspondance sont donnés par les clauses des règles syntaxiques. Un motif de correspondance est dit traduisible si chacune de ses clauses peut être traduite.

- p est un motif
- e est l'expression qui sera évaluée si le motif est vérifié

$$\begin{aligned}
\text{(PATTAG)}\quad & \mathit{Case}(\mathit{ConstructorPattern}((n, c)), e) \\
& \quad \to \mathit{MatchPatTag}(\mathit{ConsNamed}(n), [\![c]\!]_{Case}, [\![e]\!]_{Expr})
\end{aligned}$$

$$\text{(PATVAR)}\quad \mathit{Case}(\mathit{VarPattern}(x), e, l) \to \mathit{MatchPatVar}((x, l), [\![e]\!]_{Expr}, l)$$