

Chapter 1

Langage des expressions arithmétiques

1.1 Syntaxe

$E_A ::= n \mid x \mid E_A + E_A \mid E_A - E_A \mid E_A \times E_A \mid E_A / E_A$

Définition inductive des expressions arithmétiques

Définition inductive de l'ensemble E_A par un système d'inférence
Jugements : $(\mathbb{Z} \cup V \cup \{+, -, \times, /\})^*$

$$\frac{}{(\mathbb{A}_1) \frac{-}{n} (n \in \mathbb{Z})} \quad \frac{}{(\mathbb{A}_2) \frac{-}{x} (x \in V)} \\ \frac{}{(\mathbb{A}_3) \frac{a_1 \quad a_2}{a_1 + a_2}} \quad \frac{}{(\mathbb{A}_4) \frac{a_1 \quad a_2}{a_1 - a_2}} \quad \frac{}{(\mathbb{A}_5) \frac{a_1 \quad a_2}{a_1 \times a_2}} \quad \frac{}{(\mathbb{A}_6) \frac{a_1 \quad a_2}{a_1 / a_2}}$$

1.1.1 Expressions arithmétiques en OCaml

```
type 'a exp_arith =  
  | Ent of int  
  | Var of 'a  
  | Plus of 'a exp_arith * 'a exp_arith  
  | Moins of 'a exp_arith * 'a exp_arith  
  | Fois of 'a exp_arith * 'a exp_arith  
  | Div of 'a exp_arith * 'a exp_arith
```

1.1.2 Expressions arithmétiques en K

```
module ARITH-SYNTAX  
  imports DOMAINS
```

```

syntax Aexp ::= Int
            | Id
            | Aexp "*" Aexp [left]
            | Aexp "/" Aexp [left]
            | Aexp "+" Aexp [left]
            | Aexp "-" Aexp [left]

endmodule

module ARITH
  imports ARITH-SYNTAX

  syntax KResult ::= Int

  configuration <T>
    <k> $PGM:K </k>
    <state> .Map </state>
  </T>

  rule N1:Int + N2:Int => N1 +Int N2
  rule N1:Int - N2:Int => N1 -Int N2
  rule N1:Int * N2:Int => N1 *Int N2
  rule N1:Int / N2:Int => N1 /Int N2 when N2 /=Int 0

  context HOLE + _
  context _:Int + HOLE
  context HOLE - _
  context _:Int - HOLE
  context HOLE * _
  context _:Int * HOLE
  context HOLE / _
  context _:Int / HOLE

  rule <k> X:Id => N ...</k><state>... X |→ N ...</state>

endmodule

```

1.1.3 Expressions arithmétiques en Coq

1.1.4 Expressions arithmétiques en Dedukti

1.2 Sémantique dénotationnelle

Nous cherchons ici à interpréter des expressions arithmétiques, c’est-à-dire donner, à chaque expression arithmétique, une valeur appartenant à $\mathbb{V} = \mathbb{Z} \cup \{Err\}$, où *Err* dénote une erreur lors de l’interprétation.

1.3. SÉMANTIQUE OPÉRATIONNELLE D'ÉVALUATION À GRANDS PAS3

Interprétation des symboles

- Interprétation des **variables** par une **valuation** : $V[\mathbb{Z}] = \{V \rightarrow \mathbb{Z}\}$ (mémoire, environnement, ...)
- Interpréter des symboles de $\mathbb{Z} \cup \{+, -, \times, /\}$:
 n est interprété par lui-même : $\llbracket n \rrbracket = n$

$v_1 \left(\begin{array}{c} \llbracket + \rrbracket \\ \llbracket - \rrbracket \\ \llbracket \times \rrbracket \end{array} \right) v_2$	$v_2 \in \mathbb{Z}$	Err
$v_1 \in \mathbb{Z}$	$v_1 \left(\begin{array}{c} + \\ - \\ \times \end{array} \right) v_2$	Err
Err	Err	Err

$v_1 \llbracket / \rrbracket v_2$	$v_2 \in \mathbb{Z} \setminus \{0\}$	0	Err
$v_1 \in \mathbb{Z}$	v_1 / v_2	Err	Err
Err	Err	Err	Err

```
type 'a valuation = 'a → int
```

1.2.1 Evaluation des expressions arithmétiques en OCaml

```
let rec eval_arith (e : 'a exp_arith) (v: 'a valuation) : valeur = match e with
| Ent n → Z n
| Var x → Z (v x)
| Plus(e1, e2) → (<+>) (eval_arith e1 v) (eval_arith e2 v)
| Moins(e1, e2) → (<->) (eval_arith e1 v) (eval_arith e2 v)
| Foix(e1, e2) → (<*>) (eval_arith e1 v) (eval_arith e2 v)
| Div(e1, e2) → (</>) (eval_arith e1 v) (eval_arith e2 v)
```

1.2.2 Expressions arithmétiques en OCaml

```
type valeur = Z of int | Err
```

1.3 Sémantique opérationnelle d'évaluation à grands pas

Déterminisme

Proposition :

Équivalence des sémantiques opérationnelle et dénotationnelle

Propriétés Évaluation des expressions arithmétiques : variables

Expressions arithmétiques équivalentes (1)

Caractériser les expressions arithmétiques qui s'évaluent à la même valeur.

Proposition : est une congruence.

1.4 Sémantique opérationnelle d'évaluation à petits pas

déterministe

1.5 Contexte d'évaluation

Équivalence des sémantiques ?