Programmation Fonctionnelle
Arbres de Syntaxe Abstraite (AST)

Adrien Durier

26 septembre 2023

AST : Expressions arithmétiques

Définition d'un type `t` pour les AST représentant les expressions arithmétiques constituées uniquement de constantes entières (type `int`) et d'opérateurs arithmétiques `+`, `-`, `*` et `/` :

```
type op = Moins | Plus | Mult | Div
type t =
```

AST : Expressions arithmétiques

Définition d'un type `t` pour les AST représentant les expressions arithmétiques constituées uniquement de constantes entières (type `int`) et d'opérateurs arithmétiques `+`, `-`, `*` et `/` :

```
type op = Moins | Plus | Mult | Div
type t =
  | Cst of int
  | MoinsUnaire of t
  | OpBinaire of ???
```

AST : Expressions arithmétiques

Définition d'un type `t` pour les AST représentant les expressions arithmétiques constituées uniquement de constantes entières (type `int`) et d'opérateurs arithmétiques `+`, `-`, `*` et `/` :

```
type op = Moins | Plus | Mult | Div
type t =
  | Cst of int
  | MoinsUnaire of t
  | OpBinaire of t * op * t
```

AST : Expressions arithmétiques

L'expression `2 * (-4 + 5)` est représentée par l'AST :

```
let e1 = OpBin(
        Cst(2),
        Mult,
        OpBin(Cst(-4), Plus, Cst(5)))
```

L'expression `-((3 - 4) * (2 + 0))` est représentée par :

```
let e2 = Minus(OpBin(
        OpBin(Cst(3), Moins, Cst(4)),
        Mult,
        OpBin(Cst(2), Plus, Cst(0))))
```

AST : Affichage

```
let print_operateur op =
  match op with
  | Moins -> printf " - "
  | Plus -> printf " + "
  | Mult -> printf " * "
  | Div -> printf " / "
```

```
let rec print e =
  match e with
  | Cst(n) -> printf "%d" n
  | Minus(e') ->
     printf "-"; print e'
  | OpBin(e1, op, e2) ->
      printf "("; print e1;
      print_operateur op;
      print e2; printf ")"
```

AST : Évaluation

```
let eval_operateur op v1 v2 =
  match op with
  | Moins -> v1 - v2
  | Plus -> v1 + v2
  | Mult -> v1 * v2
  | Div -> v1 / v2
```

```
let rec eval e =
  match e with
  | Cst(n) -> ???
  | Minus(e') -> ???
  | OpBin(e1, op, e2) ->
     ???


-
```

AST : Évaluation

```
let eval_operateur op v1 v2 =
  match op with
  | Moins -> v1 - v2
  | Plus -> v1 + v2
  | Mult -> v1 * v2
  | Div -> v1 / v2
```

```
let rec eval e =
  match e with
  | Cst(n) -> n
  | Minus(e') -> ???
  | OpBin(e1, op, e2) ->
     ???


-
```

AST : Évaluation

```
let eval_operateur op v1 v2 =
  match op with
  | Moins -> v1 - v2
  | Plus -> v1 + v2
  | Mult -> v1 * v2
  | Div -> v1 / v2
```

```
let rec eval e =
  match e with
  | Cst(n) -> n
  | Minus(e') -> - (eval e')
  | OpBin(e1, op, e2) ->
     ???


-
```

AST : Évaluation

```
let eval_operateur op v1 v2 =
  match op with
  | Moins -> v1 - v2
  | Plus -> v1 + v2
  | Mult -> v1 * v2
  | Div -> v1 / v2
```

```
let rec eval e =
  match e with
  | Cst(n) -> n
  | Minus(e') -> - (eval e')
  | OpBin(e1, op, e2) ->
     let v1 = eval e1 in
     let v2 = eval e2 in
     eval_operateur op v1 v2
```