Programmation Fonctionnelle : des concepts aux applications web (NFP119)

Syntaxe abstraite

Tristan Crolard

Laboratoire CEDRIC
Equipe « Systèmes Sûrs »

tristan.crolard@cnam.fr

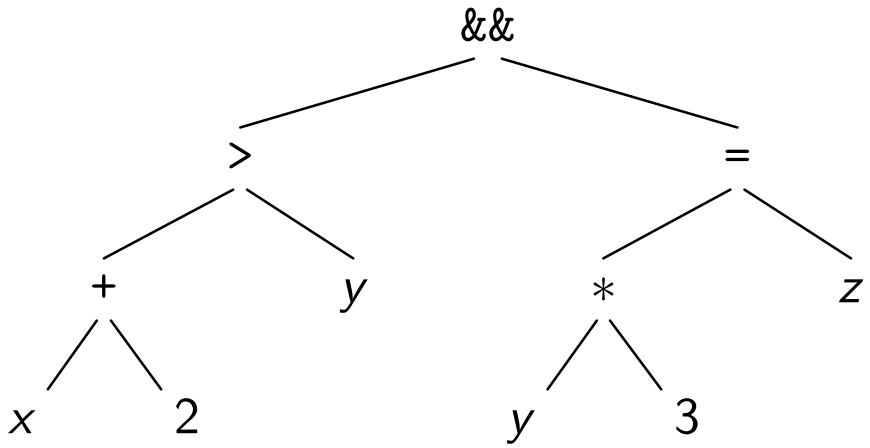
cedric.cnam.fr/sys/crolard

Expressions arithmétiques et booléennes

Exemple : syntaxe concrète

$$(x + 2) > y \;\&\&\; y * 3 = z$$

Exemple : syntaxe abstraite



Grammaire

$$
\begin{array}{rcl}
e & ::= & \\
 & | & q \\
 & | & x \\
 & | & \text{true} \\
 & | & \text{false} \\
 & | & e_1 \text{ + } e_2 \\
 & | & e_1 \text{ - } e_2 \\
 & | & e_1 * e_2 \\
 & | & e_1 < e_2 \\
 & | & e_1 > e_2 \\
 & | & e_1 = e_2 \\
 & | & e_1 \text{ \&\& } e_2 \\
 & | & e_1 \text{ || } e_2 \\
 & | & \text{not } e_1
\end{array}
$$

Type de données en ML (syntaxe OCaml)

```
type exp =
        | Cst of int
        | Var of string
        | True
        | False
        | Plus of exp * exp
        | Minus of exp * exp
        | Times of exp * exp
        | Greater of exp * exp
        | Less of exp * exp
        | Equal of exp * exp
        | And of exp * exp
        | Or of exp * exp
        | Not of exp
```

1 Evaluation d'une expression arithmétique (sans variables)

$$
\begin{array}{rcl}
e & ::= & \\
 & | & q \\
 & | & e_1 \texttt{ + } e_2 \\
 & | & e_1 \texttt{ - } e_2 \\
 & | & e_1 \texttt{ * } e_2
\end{array}
$$

$$
\begin{array}{rcl}
[\![q]\!] & = & q \\
[\![e_1 + e_2]\!] & = & [\![e_1]\!] + [\![e_2]\!] \\
[\![e_1 - e_2]\!] & = & [\![e_1]\!] - [\![e_2]\!] \\
[\![e_1 \times e_2]\!] & = & [\![e_1]\!] \times [\![e_2]\!]
\end{array}
$$

2 Evaluation d'une expression arithmétique (sans variables) en ML

```
type exp =
   | Cst of int
   | Plus of exp * exp
   | Minus of exp * exp
   | Times of exp * exp


let rec eval_exp e =
   match e with
      | Cst q -> q
      | Plus (e1, e2) -> eval_exp e1 + eval_exp e2
      | Minus (e1, e2) -> eval_exp e1 - eval_exp e2
      | Times (e1, e2) -> eval_exp e1 * eval_exp e2
```

3 Evaluation d'une expression arithmétique (avec variables)

$$
\begin{array}{rcl}
e & ::= & \\
 & | & q \\
 & | & x \\
 & | & e_1 \texttt{ + } e_2 \\
 & | & e_1 \texttt{ - } e_2 \\
 & | & e_1 \texttt{ * } e_2
\end{array}
$$

$$
\begin{array}{rcl}
[\![q]\!]_\sigma & = & q \\
[\![x]\!]_\sigma & = & \sigma(x) \\
[\![e_1 + e_2]\!]_\sigma & = & [\![e_1]\!]_\sigma + [\![e_2]\!]_\sigma \\
[\![e_1 - e_2]\!]_\sigma & = & [\![e_1]\!]_\sigma - [\![e_2]\!]_\sigma \\
[\![e_1 \times e_2]\!]_\sigma & = & [\![e_1]\!]_\sigma \times [\![e_2]\!]_\sigma
\end{array}
$$

4 Evaluation d'une expression arithmétique (avec variables) en ML

```
type exp =
   | Cst of int
   | Var of string
   | Plus of exp * exp
   | Minus of exp * exp
   | Times of exp * exp

let rec eval_exp (e,s) =
   match e with
      | Cst q -> q
      | Var x -> s x
      | Plus (e1, e2) -> eval_exp (e1,s) + eval_exp (e2,s)
      | Minus (e1, e2) -> eval_exp (e1,s) - eval_exp (e2,s)
      | Times (e1, e2) -> eval_exp (e1,s) * eval_exp (e2,s)
```