Université Nice Sophia Antipolis

Paradigmes et Interprétation

Expressions arithmétiques et fonctions

Julien Provillard

julien.provillard@univ-cotedazur.fr

Université Nice Sophia Antipolis

EXPRESSIONS ARITHMÉTIQUES

Université Nice Sophia Antipolis

Expressions arithmétiques

- Premier langage simple pour introduire les outils et la méthodologie.
- Dans ce langage, on aura comme expressions :
  - L'addition `{+ 1 2}`
  - La multiplication `{* 3 4}`

Notez l'utilisation des accolades pour le langage que nous définissons.

- On peut composer des expressions :

```
{+ 1 {* 2 3}}
```

- Les arguments des opérations sont eux-mêmes des expressions, les constantes sont donc des expressions.

Université Nice Sophia Antipolis

Représentation des expressions

- Une expression est donc :
  - Un nombre
  - Une addition
    - Qui prend deux expressions en paramètre
  - Une multiplication
    - Qui prend deux expressions en paramètre

```
(define-type Exp
  [numE (n : Number)]
  [plusE (l : Exp) (r : Exp)]
  [multE (l : Exp) (r : Exp)])
```

Université Nice Sophia Antipolis

Interprétation

- On veut pouvoir interpréter une expression arithmétique, c'est-à-dire disposer d'une fonction `interp : (Exp -> Number)`

```
(define (interp [e : Exp]) : Number
  (type-case Exp e
    [(numE n) n]
    [(plusE l r) (+ (interp l) (interp r))]
    [(multE l r) (* (interp l) (interp r))]))
```

Université Nice Sophia Antipolis

Exemples

```
; 1
(interp (numE 1))
--> 1

; {+ 1 2}
(interp (plusE (numE 1) (numE 2)))
--> 3

; {* 2 3}
(interp (multE (numE 2) (numE 3)))
--> 6

; {+ 1 {* 2 3}}
(interp (plusE (numE 1) (multE (numE 2) (numE 3))))
--> 7
```

Université Nice Sophia Antipolis

Syntaxe concrète

- Il est fastidieux de devoir écrire la représentation d'une expression à chaque fois, on voudrait pouvoir passer directement l'expression.

- On peut ajouter un backquote devant une expression pour la transformer en s-expression :

```
`{+ 1 {* 2 3}} : S-Exp
```

- Une s-expression est soit un atome, soit une liste de s-expressions.
- On peut analyser une s-expression pour produire notre représentation interne.

Université Nice Sophia Antipolis

Manipulation des s-expressions

Prédicats

```
(s-exp-symbol? s)     (s-exp-string? s)
(s-exp-number? s)     (s-exp-boolean? s)
(s-exp-list? s)
```

Conversions

```
(s-exp->symbol s)     (s-exp->string s)
(s-exp->number s)     (s-exp->boolean s)
(s-exp->list s)
```

Université Nice Sophia Antipolis

Analyse syntaxique

```
(define (parse [s : S-Exp]) : Exp
  (cond
    [(s-exp-number? s) (numE (s-exp->number s))]
    [(s-exp-list? s)
     (let ([sl (s-exp->list s)])
       (if (and (= (length sl) 3) (s-exp-symbol? (first sl)))
           (let ([op (s-exp->symbol (first sl))])
             (cond
               [(equal? op '+) (plusE (parse (second sl)) (parse (third sl)))]
               [(equal? op '*) (multE (parse (second sl)) (parse (third sl)))]
               [else (error 'parse "invalid input")]))
           (error 'parse "invalid input")))]
    [else (error 'parse "invalid input")]))
```

Université Nice Sophia Antipolis

Exemples d'utilisation

```
(interp (parse '2))
--> 2

(interp (parse '{+ 1 2}))
--> 3

(interp (parse '{* 1 2}))
--> 2

(interp (parse '{+ 1 {* 2 3}}))
--> 7
```

Université Nice Sophia Antipolis

Analyse syntaxique

- Le but de l'analyseur syntaxique est de transformer une s-expression en un arbre syntaxique dans notre représentation interne.
- La fonction `parse` devra donc être redéfinie pour chaque langage ou toute modification d'un langage existant.
- Peut-on se simplifier la vie ?
- On peut faire de la reconnaissance de motifs !

Université Nice Sophia Antipolis

Reconnaissance de motifs

- La fonction `s-exp-match?` permet de faire de la reconnaissance de motifs.
- L'appel `(s-exp-match? p s)` vérifie que la s-expression `s` est conforme au motif `p` (une s-expression avec des symboles d'échappement).
- Ces symboles d'échappement sont :

Symbole	s-expression reconnue
`ANY`	N'importe quelle s-expression
`SYMBOL`	Un symbole
`NUMBER`	Un nombre
`STRING`	Une chaîne de caractères
`p ...`	Le motif p répété n'importe quel nombre de fois, y compris 0

Université Nice Sophia Antipolis

La fonction parse simplifiée

```
(define (parse [s : S-Exp]) : Exp
  (cond
    [(s-exp-match? `NUMBER s) (numE (s-exp->number s))]
    [(s-exp-match? `{+ ANY ANY} s)
     (let ([sl (s-exp->list s)])
       (plusE (parse (second sl)) (parse (third sl))))]
    [(s-exp-match? `{* ANY ANY} s)
     (let ([sl (s-exp->list s)])
       (multE (parse (second sl)) (parse (third sl))))]
    [else (error 'parse "invalid input")]))
```

Université Nice Sophia Antipolis

FONCTIONS

Université Nice Sophia Antipolis

Fonctions

- On veut enrichir notre langage d'expressions arithmétiques avec des fonctions.

```
{define {double x} {+ x x}}
{define {quadruple x} {double {double x}}}

{+ {define {double x} {+ x x}} 1}
```

- Une **définition** de fonction n'est pas une expression.

```
{+ {double 3} 1}
```

- Une **application** de fonction est une expression.

Université Nice Sophia Antipolis

Définitions de fonction

- `{define {double x} {+ x x}}`
- Une définition de fonction est composée :
  - D'un nom : `double` → représenté par un symbole
  - D'un paramètre : `x` → représenté par un symbole
  - D'un corps : `{+ x x}`

```
(define-type Body
  [numB (n : Number)]
  [idB (s : Symbol)]
  [plusB (l : Body) (r : Body)]
  [multB (l : Body) (r : Body)])
```

Très proche de `Exp`

Université Nice Sophia Antipolis

Définitions de fonction

- `{define {double x} {+ x x}}`
- Une définition de fonction est composée :
  - D'un nom : `double` ➡ représenté par un symbole
  - D'un paramètre : `x` ➡ représenté par un symbole
  - D'un corps : `{+ x x}` ➡ représenté par une expression

- En ajoutant les identificateurs au langage, on peut représenter le corps d'une fonction par une expression.

Université Nice Sophia Antipolis

Le langage

- Expressions
  - Nombres
  - Identificateurs
  - Addition
    - Prend deux expressions en arguments
  - Multiplication
    - Prend deux expressions en arguments
  - Application de fonction
    - Prend un nom de fonction et une expression en arguments
- Définition de fonctions
  - Prend un nom de fonction, un nom de paramètre et une expression en arguments

Université Nice Sophia Antipolis

Représentation

Expressions

```
(define-type Exp
  [numE (n : Number)]
  [idE (s : Symbol)]
  [plusE (l : Exp) (r : Exp)]
  [multE (l : Exp) (r : Exp)]
  [appE (fun : Symbol) (arg : Exp)])
```

Définitions de fonctions

```
(define-type FunDef
  [fd (name : Symbol) (par : Symbol) (body : Exp)])
```

Université Nice Sophia Antipolis

Représentation

- `{define {double x} {+ x x}}`

```
(fd 'double 'x
    (plusE (idE 'x) (idE 'x)))
```

- `{define {quadruple x} {double {double x}}}`

```
(fd 'quadruple 'x
    (appE 'double
          (appE 'double (idE 'x))))
```

Université Nice Sophia Antipolis

Comment évaluer une fonction?

- `{define {double x} {+ x x}}`
- Comment évaluer l'application `{double 3}` ?
  - On cherche le corps de la fonction `double`

    `{+ x x}`
  - On substitue l'argument pour chaque occurrence du paramètre formel

    `{+ 3 3}`
  - On évalue le corps

    `6`

Université Nice Sophia Antipolis

Impact sur l'interpréteur

- La fonction `interp` doit connaître toutes les définitions de fonctions.

```
interp : (Exp (Listof FunDef) -> Number)
```

- On doit pouvoir rechercher une définition de fonction par son nom.

```
get-fundef : (symbol (Listof FunDef) -> FunDef)
```

- On doit pouvoir substituer dans une expression.

```
subst : (Exp Symbol Exp -> Exp)
```

- L'appel `(subst what for in)` remplace toutes les occurences de `for` par `what` dans `in`.

```
(subst {+ 1 2} x {* x {+ x y}})
--> {* {+ 1 2} {+ {+ 1 2} y}}
```

Université Nice Sophia Antipolis

Evaluation des fonctions

```
(define (interp [e : Exp] [fds : (Listof FunDef)]) : Number
  (type-case Exp e
    [numE (n) n]
    [idE (s) (error 'interp "free identifier")]
    [plusE (l r) (+ (interp l fds) (interp r fds))]
    [multE (l r) (* (interp l fds) (interp r fds))]
    [appE (f arg) (let [(fd (get-fundef f fds))]
                    (interp (subst arg
                                   (fd-par fd)
                                   (fd-body fd))
                            fds))]))
```

Université Nice Sophia Antipolis

Evaluation des fonctions

- Est-ce qu'on aurait pu faire mieux ?

```
{define {double x} {+ x x}}
{define {quadruple x} {double {double x}}}
```

Lors de l'évaluation de `{quadruple 3}` on effectue ces étapes :

```
    {quadruple 3}
--> {double {double 3}}
--> {+ {double 3} {double 3}}
--> {+ {+ 3 3} {+ 3 3}}
--> 12
```

- On aurait pu avoir une meilleure stratégie :

```
{double {double 3}} --> {double 6} --> 12
```

Université Nice Sophia Antipolis

Evaluation des fonctions

```
(define (interp [e : Exp] [fds : (Listof FunDef)]) : Number
  (type-case Exp e
    [numE (n) n]
    [idE (s) (error 'interp "free identifier")]
    [plusE (l r) (+ (interp l fds) (interp r fds))]
    [multE (l r) (* (interp l fds) (interp r fds))]
    [appE (f arg) (let [(fd (get-fundef f fds))]
                    (interp (subst (interp arg fds)
                                   (fd-par fd)
                                   (fd-body fd))
                            fds))]))
```

Ce n'est plus une expression !

Université Nice Sophia Antipolis

Evaluation des fonctions

```
(define (interp [e : Exp] [fds : (Listof FunDef)]) : Number
  (type-case Exp e
    [numE (n) n]
    [idE (s) (error 'interp "free identifier")]
    [plusE (l r) (+ (interp l fds) (interp r fds))]
    [multE (l r) (* (interp l fds) (interp r fds))]
    [appE (f arg) (let [(fd (get-fundef f fds))]
                    (interp (subst (numE (interp arg fds))
                                   (fd-par fd)
                                   (fd-body fd))
                            fds))]))
```

Université Nice Sophia Antipolis

Les fonctions utilitaires

```
(define (get-fundef [s : Symbol] [fds : (Listof FunDef)]) : FunDef
  (cond
    [(empty? fds) (error 'get-fundef "undefined function")]
    [(equal? s (fd-name (first fds))) (first fds)]
    [else (get-fundef s (rest fds))]))

(define (subst [what : Exp] [for : Symbol] [in : Exp]) : Exp
  (type-case Exp in
    [numE (n) in]
    [idE (s) (if (equal? s for) what in)]
    [plusE (l r) (plusE (subst what for l) (subst what for r))]
    [multE (l r) (multE (subst what for l) (subst what for r))]
    [appE (f arg) (appE f (subst what for arg))]))
```

Université Nice Sophia Antipolis

Analyse syntaxique des fonctions

```
(define (parse-fundef [s : S-Exp]) : FunDef
  (if (s-exp-match? `{define {SYMBOL SYMBOL} ANY} s)
      (let ([sl (s-exp->list s)])
        (let ([sl2 (s-exp->list (second sl))])
          (fd (s-exp->symbol (first sl2))
              (s-exp->symbol (second sl2))
              (parse (third sl)))))
      (error 'parse-fundef "invalid input")))
```

Université Nice Sophia Antipolis

Exemples

```
(define (interp-expr [e : S-Exp] [fds : (Listof S-Exp)]) : Number
  (interp (parse e) (map parse-fundef fds)))

(interp-expr `{double 3}
             (list `{define {double x} {+ x x}}))

(interp-expr `{quadruple 3}
             (list `{define {double x} {+ x x}}
                   `{define {quadruple x} {double {double x}}}))
```