Univ rsité Paris Did rot — Programmation Fonctionn ll
L2 Math-Info/L3 Informatiqu /M1 Linguistiqu — Anné 2011-2012

TP n 5

XML et expressions symboliques

L'obj ctif d c TP st doubl . En pr mièr parti compr ndr la structur d docum nt XML t l s manipulations d bas à l'aid d la bibliothèqu `Xml-Light`. En d uxièm parti , manipul r d s xpr ssions mathématiqu s à plusi urs *symboles*, pour l s évalu r t l s simplifi r.

1 Définition de XML

XML st n fait un méta-langag [1] à balis p rm ttant d définir d s formats d docum nts t d cré r d s docum nts r sp ctant c s formats. Ainsi, t c' st important, XML n' st pas un format d docum nt comm t l. L s docum nts XML s compos nt d'unités d stockag app lé s *entités*, qui conti nn nt d s donné s analysabl s ou non. L s donné s analysabl s s compos nt d caractèr s, c rtains formant l s donné s t xtu ll s, t l r st formant l *balisage*. L balisag décrit l s structur s logiqu s t d stockag du docum nt. XML fournit un mécanism pour impos r d s contraint s à c s structur s.

Exemple de structure XML :

```
<?xml version="1.0" encoding="UTF-8"?>
<c t logue>
    <!--ceci est un comment ire -->
    <f mille n="f1">
        <ligne n="l11">
             <coef>2</coef>
             <GRAPH> v+ ro0  v+</GRAPH>
        </ligne>
    </f mille>
</c t logue>
```

R marqu s :
– `c t logue` st l'élém nt racin
– dans `<f mille n="f1">`, `n` st un attribut d val ur "f1"
– dans `<coef>2</coef>`, 2 st du cont nu t xt
– un élém nt = balis d'ouv rtur + cont nu + balis d f rm tur
– l cont nu d'un élém nt = un nom + ns mbl d'attributs + ns mbl d'élém nts.

Parseur XML : L pars ur p rm t d'un part d' xtrair l s donné s d'un docum nt XML (on parl d'analys du docum nt ou d parsing) ainsi qu d vérifi r év ntu ll m nt la validité du docum nt.

1. c'est-à-dire qui permet de créer de nouveaux langages, comme par exemple XHTML

2 Xml-Light

Xml-Light st un pars ur XML pour OCaml. Il fournit d s fonctions pour transform r un document XML n un structur d donné s OCaml, la manipul r, t év ntu ll m nt génér r un document XML n sorti . Il p ut égal m nt aid r à vérifi r la validité d'un document vis-à-vis d'un DTD.

Structure xml en OCaml : un nœud XML st soit un El m nt av c d s argum nts d la form $(tag\_name, attributes, children)$, soit un PCData suivi d'un t xt .

```
type xml =
 | Element of (string * (string * string) list * xml list)
 | PCD t  of string
```

Fonctions de base :

```
(* Tr nsforme un fichier XML en données XML. *)
v l p rse_file : string -> xml

(* Tr nsforme en données XML une ch îne conten nt un document XML. *)
v l p rse_string : string -> xml

(* Affiche les données XML en ch ine de c r ctères de f çon comp cte. *)
v l to_string : xml -> string
```

Exemple :

```
#directory "+xml-light";;
#lo d "xml-light.cm ";;
let x = Xml.p rse_file " gents.xml" in
    print_endline (Xml.to_string x)
```

L s d ux pr mièr s instructions charg nt la librairi Xml-Light. C programm pars l docum nt *agents.xml* n typ OCaml `Xml.xml` t affich nsuit l résultat n chain s d caractèr s.

3 Définition d'une expression symbolique

Un *expression à plusieurs symboles*, ou *expression symbolique*, st un obj t mathématiqu qui p ut êtr :
– soit un nombr flottant
– soit l'un d s form s suivant s : $C_1 + C_2$, $C_1 \quad C_2$, $C_1 \quad C_2$, $\cos C_1$, $\sin C_2$, où C_1 t C_2 sont d ux xpr ssions symboliqu s
– soit un *symbole* (x, y, ab, tc.)
Par x mpl :

$$((2. \quad 1.) \quad (1. \quad x)) \quad ((19. \quad \cos(0.)) + \sin(theta))$$

st un xpr ssion qui conti nt d ux symbol s, x t *theta*.

Exercice 1 On donn l s typ s suivants pour r prés nt r l s opérat urs binair s t unair s :

```
type binop = Plus | Moins | Fois
type unop  = Cos  | Sin
```

Définir le type `expr` pour représenter les expressions symboliques, sous la forme d'un type somme à quatre constructeurs : `Nombre`, `Binop`, `Unop` et `Symbole` (un symbole étant représenté par une chaîne de caractères), de sorte que l'expression e_0 ci-dessus soit représentée en OCaml par :

```
let e0 = Binop (Fois,
                Binop (Moins,
                       Binop (Fois,  Nombre 2.,  Nombre 1.),
                       Binop (Fois,  Nombre 1.,  Symbole "x")),
                Binop (Plus,
                       Binop (Moins, Nombre 19., Unop (Cos, Nombre 0.)),
                       Unop  (Sin,   Symbole "thet "))) ;;
```

Exercice 2 Écrire une fonction :

```
string_of_expr : expr -> string
```

qui, étant donnée une expression, renvoie une chaîne de caractères représentant l'expression entièrement parenthésée sous forme lisible par un être humain. Par exemple, pour e_0, cette fonction renverra la chaîne :

```
"(((2.*1.)-(1.*x))*((19.-cos(0.))+sin(thet )))"
```

On pourra écrire au préalable des fonctions :
– `string_of_binop : binop -> string`
– `string_of_unop : unop -> string`
qui étant donné un opérateur, renvoient sa représentation lisible en chaîne de caractères.

Exercice 3 Un intérêt d'utiliser XML est de structurer des expressions symboliques de type `expr` en document pouvant être stocké, transmis entre des programmes, envoyé à travers un réseau, etc. Écrire une fonction :

```
expr_of_xml : Xml.xml -> expr
```

qui lit une expression sous forme XML (correspondant par exemple au document *expr.xml* ci-dessous) et retourne une expression de type `expr`.

```
<?xml version="1.0"  encoding="UTF-8" ?>
<Binop>
  <Moins/>
  <Binop>
    <Fois/>
    <Nombre>2</Nombre>
    <Nombre>1</Nombre>
  </Binop>
  <Binop>
    <Fois/>
    <Nombre>1</Nombre>
    <Symbole>"X"</Symbole>
  </Binop>
</Binop>
```

4 É aluation d'une expression symbolique

Exercice 4 Écrir un fonction :

```
ev l_expr : (string * flo t) list -> expr -> flo t
```

t ll qu , si `l` st un *environnement* associant (sous la form d'un list d'association) chaqu symbol à un val ur flottant , t si `e` st un xpr ssion, alors `ev l_expr l e` calcul t r nvoi la val ur d l' xpr ssion e sous l' nvironn m nt l. Par x mpl , pour évalu r l' xpr ssion e_0 ci-d ssus n pr nant 3. pour x t 0. pour *theta*, on écrira :

```
ev l_expr [("x", 3.); ("thet ", 0.)] e0
```

c qui r nv rra `-18.` (car $\cos 0 = 1$ t $\sin 0 = 0$).

Indications : On pourra utilis r la fonction `List. ssoc : ’ -> (’ * ’b) list -> ’b`.
On pourra aussi écrir au préalabl d s fonctions :
– `v leur_binop : binop -> flo t -> flo t -> flo t`
– `v leur_unop : unop -> flo t -> flo t`
qui ff ctu nt l s opérations binair s t unair s n supposant qu l urs argum nts sont d s val urs flottant s déjà calculé s.

5 Simplification d'une expression symbolique

Exercice 5 Écrir un fonction :

```
ev l_sous_expr : expr -> expr
```

t ll qu , si e st un xpr ssion, alors `ev l_sous_expr e` r nvoi l' xpr ssion obt nu n r mplaçant tout sous- xpr ssion d e n cont nant pas d symbol s, par sa val ur. C tt pass st app lé *évaluation des sous-expressions constantes.*
Par x mpl , dans l' xpr ssion e_0 ci-d ssus, l s sous- xpr ssions 2. 1. t 19. cos 0. n conti nn nt pas d symbol s, ll s pourront donc êtr r mplacé s par l urs val urs, t alors `ev l_expr_p rtielle e0` r nv rra l' xpr ssion : $(2. \quad (1. \quad x)) \quad (18. + \sin(theta))$
Fair d s ssais pour r marqu r qu , si e st un xpr ssion qu lconqu , alors, pour tout nvironn m nt l, `ev l_expr l e` t `ev l_expr l (ev l_sous_expr e)` r nvoi nt l s mêm s val urs.
On dit qu `ev l_sous_expr` st *correcte* vis-à-vis d l'évaluation d s xpr ssions.
R marqu r aussi qu pour tout xpr ssion e :

$$\texttt{ev l\_sous\_expr}(\texttt{ev l\_sous\_expr}\ e) = \texttt{ev l\_sous\_expr}\ e$$

On dit qu `ev l_sous_expr` st *idempotente.*

Exercice 6 Écrir un fonction :

```
ev l_neutres : expr -> expr
```

t ll qu , si e st un xpr ssion, alors `ev l_neutres e` r nvoi l' xpr ssion obt nu n r mplaçant tout sous- xpr ssion d e :
– d la form $0. \quad e'$ ou $e' \quad 0.$ ou $e' \quad e'$ par 0.
– d la form $1 \quad e', e' \quad 1$ ou $e' \quad 0$ ou $e' + 0.$ ou $0. + e'$ par e'

Par x mpl , dans l' xpr ssion e_0 ci-d ssus, la sous- xpr ssion 2 1 (r sp. 1 x) pourra êtr r mplacé par 2 (r sp. x), t alors `ev l_neutres e0` r nv rra l' xpr ssion :

$$(2. \quad x) \quad ((19. \quad \cos(0.)) + \sin(theta))$$

(l'évaluation d s sous- xpr ssions constant s n'a pas li u)
R marqu r qu `ev l_neutres` st corr ct vis-à-vis d l'évaluation d s xpr ssions, t qu' ll st id mpot nt .

Exercice 7 Écrir un fonction :

```
point_fixe : (' -> ' ) -> ' -> '
```

t ll qu , si f st un fonction t si a st un val ur applicabl à f, alors `point_fixe f` appliqu f sur a t r comm nc sur l résultat jusqu'à c qu'il n chang plus.
On dit qu `point_fixe f` st un *point fixe* d f (n ff t, si $x =$ `point_fixe f` , alors `f` $x = x$). C' st l point fix d f obt nu n partant d a.
Utilis r c tt fonction pour écrir un fonction :

```
simplifier : expr -> expr
```

qui, étant donné un xpr ssion, lui appliqu l s d ux fonctions `ev l_neutres` t `ev l_sous_expr` t continu jusqu'à c qu l' xpr ssion n chang plus.
Pourquoi c tt fonction t rmin -t- ll ?