Arbres et expressions

Syntaxe abstraite

L'ensemble $\mathcal{E}$ des expressions arithmétiques se définit de façon récursive.

On considère un ensemble $\mathcal{C}$ de *constantes*, un ensemble $\mathcal{V}$ de *variables*, un ensemble fini $\mathcal{O}$ d'opérateurs binaires et un ensemble fini $\mathcal{F}$ d'opérateurs unaires ou *fonctions*. Par exemple : $\mathcal{C} = \mathbb{R}$, $\mathcal{V} = \{x_1, x_2, \ldots\}$, $\mathcal{O} = \{+, -, \times, /\}$ et $\mathcal{F} = \{\sin, \cos, \tan, \sqrt{}, \ln\}$.

Alors toute constante est une expression, toute variable est une expression, et, si $c \in \mathcal{O}$ et $f \in \mathcal{F}$ et e_1 et e_2 sont deux expressions, $(e_1\ c\ e_2)$ et $(f\ e_1)$ sont des expressions.

En pratique, les règles usuelles de priorité entre opérateurs et d'associativité permettent de réduire le nombre de parenthèses utiles.

Ainsi, on écrira $\sin(\pi/4) + 3 \times \cos(2 \times \pi/5)$ et non pas

$$((\sin(\pi/4)) + (3 \times (\cos(2 \times (\pi/5))))).$$

La *grammaire* des expressions peut donc être écrite ainsi :

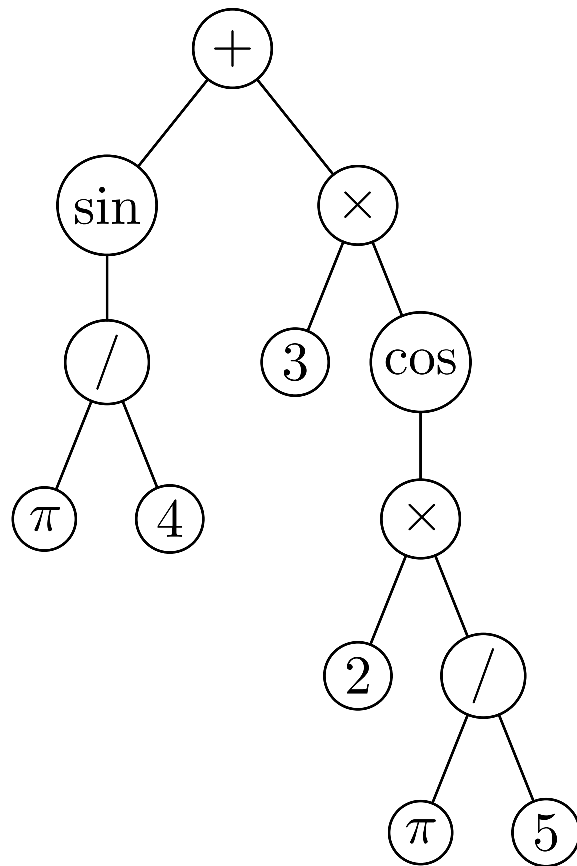
expression	::=	constante
	|	variable
	|	(expression op expression)
	|	(fonction expression)

Syntaxe concrète : arbres d'expression

On associe naturellement à une expression arithmétique un arbre général : aux variables et constantes correspondent les feuilles de l'arbre, aux opérateurs binaires des nœuds binaires, et aux fonctions des nœuds unaires.

Par exemple, voici page suivante l'arbre de l'expression

$$\sin(\pi/4) + 3 \times \cos(2 \times \pi/5).$$


+
sin
×
/
3
cos
π
4
×
2
/
π
5

Syntaxe concrète : Caml

Le typage Caml correspondant est immédiat :

```
type ('c,'v,'o,'f) expr =
    | Constante of 'c
    | Variable of 'v
    | Terme2 of ('c,'v,'o,'f) expr * 'o  * ('c,'v,'o,'f) expr
    | Terme1 of 'f * ('c,'v,'o,'f) expr ;;
```

mais comme en pratique nos opérateurs binaires sont tous associatifs à gauche, on préfèrera :

```
type ('c,'v,'o,'f) expression =
    | Constante of 'c
    | Variable of 'v
    | Terme of 'o * ('c,'v,'o,'f) expression list
    | Applique of 'f * ('c,'v,'o,'f) expression ;;
```

et par exemple on définira

```
type fonction = Sin | Cos | Tan | Sqrt | Ln ;;
type expr == (float,string,char,fonction) expression ;;
```

On a choisi de représenter les variables par leurs noms, qui sont des chaînes de caractères ; et les opérateurs binaires par leur symbole, qui est un caractère.

On représentera par exemple l'expression $1.3 + \sqrt{2} + x$ par :

```
let exemple = Terme(`+`,[Constante 1.3 ;
                         Applique(Sqrt,Constante 2.0) ;
                         Variable "x"]) ;;
```

Sémantique des expressions

Notion de contexte Un *contexte (d'évaluation)* est simplement une application φ de $\mathcal{V}$, l'ensemble des variables, dans $\mathcal{C}$, l'ensemble des valeurs. Cependant on notera $[\varphi]\,v$ au lieu de $\varphi(v)$, ce qui se justifiera bientôt.

En Caml, un contexte est souvent représenté par une liste de couples (v, c) variable-valeur, c'est-à-dire une liste associative de type `('v * 'c) list` et on écrira souvent un contexte sous cette forme : $[(v_1, c_1); (v_2, c_2); \ldots]\,v = c_k$ dès que $v = v_k$.

Dans la suite, l'ensemble des contextes est (naturellement) noté $\mathcal{C}^{\mathcal{V}}$.

À tout opérateur binaire o on associe son interprétation $\tilde{o} : \mathcal{C} \times \mathcal{C} \longrightarrow \mathcal{C}$ et, de même, à toute fonction f on associe son interprétation $\tilde{f} : \mathcal{C} \longrightarrow \mathcal{C}$.

Ainsi au **symbole** sin on associe son interprétation : le sinus, etc.

On définit alors une *sémantique* en introduisant la fonction d'évaluation définie sur $\mathcal{C}^{\mathcal{V}} \times \mathcal{E}$ de la façon suivante :

- $[\varphi]\, c = c$ pour tout contexte φ et toute constante c ;
- $[\varphi]\, v = \varphi(v)$ pour tout contexte φ et toute variable v ;
- $[\varphi]\, (e_1\; o\; e_2) = \tilde{o}([\varphi]\, e_1, [\varphi]\, e_2)$, pour tout contexte φ, toutes expressions e_1 et e_2, et tout opérateur o d'interprétation $\tilde{o}$;
- $[\varphi]\, (f\; e) = \tilde{f}([\varphi]\, e)$, pour tout contexte φ, toute expression e et toute fonction f d'interprétation $\tilde{f}$.

On traduit ceci immédiatement en CAML :

```
let rec assoc v = function
    | [] -> failwith "Contexte incomplet"
    | (w,x) :: q when w = v -> x
    | _ :: q -> assoc v q
and compose f (t :: q) = match q with
    | [] -> t
    | t' :: q' -> compose f ((f t t') :: q) ;;

let rec eval contexte = function
    | Constante x -> x
    | Variable v -> assoc v contexte
    | Terme(`+`,l) -> compose (fun x y -> x +. y) (map (eval contexte) l)
    | Terme(`-`,l) -> compose (fun x y -> x -. y) (map (eval contexte) l)
    | Terme(`*`,l) -> compose (fun x y -> x *. y) (map (eval contexte) l)
    | Terme(`/`,l) -> compose (fun x y -> x /. y) (map (eval contexte) l)
    | Applique(f,e) -> let x = eval contexte e in match f with
                       | Sin -> sin x | Cos -> cos x | Tan -> tan x
                       | Sqrt -> sqrt x | Ln -> log x ;;
```

On laisse en exercice au lecteur l'écriture d'une fonction
`dérive : expr -> string -> expr`

Ce n'est pas très compliqué, et même plutôt amusant !

Là où cela se compliquerait, c'est si l'on demandait d'écrire une fonction de **simplification** des expressions, problème difficile qui dépasse très largement le cadre et les ambitions de ce cours.