Fondements de la programmation

Exercices 0 Grammaires (informelles)

Voici une grammaire utilisée dans ce TD :

Les tokens.

$$\begin{array}{ll} \textsc{Variables} & x = [a-z]^+ \\ \textsc{Numbers} & n = [0-9]^+ \\ \textsc{Strings} & s = \texttt{"}[\sim \texttt{"}]^*\texttt{"} \end{array}$$

Les non terminaux. les noms *id* (comme les noms de variables) sont laissés au choix parmi de nouveaux noms.

Type	$\tau ::=$	`Num`	(nombres)
		`Str`	(textes)
		$\tau_1 \to \tau_2$	(fonctions)
		$(\tau_1, \ldots, \tau_n)$	(produit)
		`Unit`	(produit vide)

Expresion	$e ::=$	x	(variable)
		n	(nombre littéral)
		s	(texte littéral)
		$e_1 + e_2$	(ajouter)
		$e_1 * e_2$	(multiplier)
		$e_1 \,/\, e_2$	(diviser)
		$e_1 \,\hat{}\, e_2$	(concaténer)
		$(e_1, \ldots, e_k)$	(tuple)
		`unit`	(tuple vide)
		`let` x`=`e_1 `in` e_2	(définition)
		$e_1\ e_2$	(application)
		`fun` $(x{:}\tau)$ $\Rightarrow$ e	(abstraction)
		`if` $e_1 \geq$ `0 then` e_2 `else` e_3	(test)

1 Exercices

1.1 Arbres

Question A. Arbre syntaxique. Écrire l'arbre syntaxique de des expressions suivante :

- `let x = "hello" in x^x`
- `fun (f : Str → Num) ⇒ f "test"`

Question B. AST. Écrire un AST des expressions suivante :

- `let toto = "hello" in x*x-32*x-79`
- `fun (f : Str → Num → Num) ⇒ f "test" 3`

Question C. Associativité/priorité. Explicitez les associativités et priorités manquantes

1.2 Étendre la grammaire

Question D. Petits points. Les ... ne seraient pas acceptés dans une grammaire formelle, trouvez une manière d'exprimer le produit formellement.

Question E. Booléens. Le test n'est pas vraiment correcte : on ne peut faire que le test à 0. Ajouter les booléens, quelques opérateurs, et un test correcte.

Question F. Les let in. Les définitions, en OCaml, sont plus générales que ça : on peut remplacer le x par un patter, qui est soit une variable, soit un tuple de patern, soit une unité, soit un underscore. Corrigez le `let_in`.