Opérateurs binaires à notation infixe en OCaml

En OCaml, les opérateurs arithmétiques usuels comme `+`,`*`,`/`,`mod` ...ou `+.`,`*.`,`/.` ..., les opérateurs booléens `&&` et `||` ou encore les opérateurs de concaténation[a] `@` et `^` sont en fait des fonctions de deux arguments qui ont la particularité de pouvoir être appelées selon la notation infixe, c'est-à-dire en étant placé entre leur deux arguments et non à gauche de ceux-ci. Afin de désigner de telles fonctions en OCaml, on encadre l'opérateur de parenthèses. Par exemple `(+)` est une expression de type fonctionnel `int -> int -> int`, alors que l'expression `+` n'est pas syntaxiquement correcte. De même :

- `(mod)` est de type `int -> int -> int` ;
- `(+.)` est de type `float -> float -> float` ;
- `(&&)` est de type `bool -> bool -> bool` ;
- `(^)` est de type `string -> string -> string` ;
- `( |> )` est de type `'a -> ('a -> 'b) -> 'b` ;
- ...[b]

De tels opérateurs ne sont pas réservés à la librairie standard, on peut définir de tels opérateurs à condition de choisir un nom de fonction valide. La description précise des noms valides est donnée dans la documentation : `https://v2.ocaml.org/manual/expr.html`, rubrique *Operators*. Sans être exhaustif, avec $\Sigma_1 = \{\texttt{\$}, \texttt{\&}, \texttt{*}, \texttt{+}, \texttt{-}, \texttt{/}, \texttt{=}, \texttt{>}, \texttt{\^{}}, \texttt{|}, \texttt{\%}, \texttt{<}\}$ et $\Sigma_2 = \Sigma_1 \cup \{\texttt{\~{}}, \texttt{!}, \texttt{?}, \texttt{:}, \texttt{.}\}$, les noms dans $\Sigma_1 \cdot \Sigma_2^*$ sont valides.

Exemple. On pourrait par exemple définir l'opérateur `%%` pour le produit scalaire sur les vecteurs d'entiers encodés par des tableaux comme suit.

```
exception ProduitScalaireImpossible
let ( %% ) (x: int array) (y: int array) : int =
  if Array.length x = Array.length y
  then
    let res = ref 0 in
    for i = 0 to Array.length x - 1 do
      res:= !res + x.(i) * y.(i)
    done; !res
  else raise ProduitScalaireImpossible
```

Après une telle définition le jeu de tests suivant est vérifié.

```
  assert ([|0; 0; 0|]    %% [|1; 2; 3|]    = 0 );
  assert ([|0; 1; 0|]    %% [|1; 2; 3|]    = 2 );
  assert ([|1; 1; 1; 1|] %% [|1; 2; 3; 4|] = 10)
```

a. Attention le symbole `::` n'est pas un opérateur de concaténation, en effet c'est un constructeur du type `'a list` (même s'il a un statut particulier qui lui permet d'être placé entre ses paramètres).

b. Les seules exceptions sont `( * )` et `( *. )` qu'il faut écrire en laissant un espace entre la parenthèse et l'étoile car l'enchaînement `(*` est réservé pour marquer un début de commentaire.