

Cours 9 : Modules paramétrés / Foncteurs

1 Définition de foncteurs

En OCaml, il est possible de définir un module paramétré par un ou plusieurs modules. On appelle de tels modules des *foncteurs*, et on peut les définir à l'aide du mot-clé **functor**. Ainsi, la définition suivante définit un foncteur M paramétré par un module A de signature S :

```
module type S = sig type t end;;  
module M = functor (A : S) -> struct type t = A.t list end;;
```

On note que le corps du foncteur M peut faire référence au module A passé en paramètre. Il existe du sucre syntaxique pour définir M de façon plus compacte et plus lisible :

```
module M' (A : S) = struct type t = A.t list end;;
```



Le module produit par un foncteur peut lui-même être un foncteur. En particulier, un foncteur peut avoir plusieurs arguments et peut être défini directement en utilisant plusieurs fois **functor**, ou en utilisant le sucre syntaxique :

```
module N = functor (A : S) ->  
    functor (B : S) -> struct type t = (A.t * B.t) list end;;  
module N' (A : S)(B : S) = struct type t = (A.t * B.t) list end;;
```



Contrairement aux fonctions standards, il n'y a pas d'inférence de type pour les paramètres des foncteurs : autrement dit, il est obligatoire de spécifier la signature des arguments d'un foncteur.

2 Application de foncteurs

Afin d'utiliser un foncteur, on peut l'appliquer (on dit aussi *instancier*) à des modules vérifiant la bonne signature. Cette application crée un module qui est le résultat du foncteur où les paramètres sont remplacés par les arguments passés au foncteur :

```
module A1 : S = struct type t = int end  
module A2 = struct  
    type t = string * bool  
    exception E of t  
end  
  
module M1 = M(A1);; (* M1.t est int list *)  
module M2 = M(A2);; (* M2.t est (string * bool) list *)
```



Notez qu'il n'est pas requis que le module passé en paramètre ait exactement la signature S , mais qu'il suffit qu'il vérifie au moins la signature S . Ainsi, l'application du foncteur M au module $A2$ est bien typée.

L'application d'un foncteur à plusieurs arguments se fait de manière similaire :

```
module N12 = N(A1)(A2);; (* N12.t est (int * (string * bool)) list *)
```

3 Signatures paramétrées

De même que les modules ont des signatures, les foncteurs ont des *signatures paramétrées*. Une signature paramétrée se définit de manière similaire à une signature standard, en utilisant encore le mot-clé `functor`. Ainsi, la signature du module M ci-dessus s'écrit de la manière suivante :

```
module type MS = functor (A : S) -> sig type t = A.t list end;;
```

Cette signature paramétrée MS est la plus précise que vérifie le foncteur M . En effet, similairement aux modules standards, OCaml attribue par défaut à un foncteur sa signature la plus générale. On peut vouloir une signature particulière afin de cacher des éléments ou de rendre des types abstraits, ainsi MS vérifie aussi toutes les signatures suivantes :

```
module type AbstractMS = functor (A : S) -> sig type t end;;  
module type AbstractMS' = functor (A : S) -> S;;  
module type ReallyAbstractMS = functor (A : S) -> sig end;;
```

Étant donné un foncteur et une signature paramétrée, on peut restreindre le foncteur à cette signature de la même manière que pour les modules standards :

```
module AbstractM : AbstractMS = M;;
```

Enfin, on peut également restreindre la signature d'un foncteur au moment de sa définition, ainsi on aurait pu définir le module $AbstractM$ directement de la façon suivante :

```
module AbstractM' (A : S) : S = struct type t = A.t list end;;
```