

1 Types de données

Types élémentaires

Haskell fournit un grand nombre de types élémentaires, dont les plus importants sont résumés dans ce tableau :

Type	Description	Bits	Bounded	Floating	Fractional	Integral	Num	Real
Double	Virgule flottante, double précision		■	■			■	■
Float	Virgule flottante, simple précision		■	■			■	■
Int	Entier signé à précision fixe, intervalle minimum $[-2^{29}; 2^{29} - 1]$							
Int8	Entier signé de 8 bits							
Int16	Entier signé de 16 bits							
Int32	Entier signé de 32 bits							
Int64	Entier signé de 64 bits							
Integer	Arbitrary precision signed integer							
Rational ou Ratio a	Nombre rationnel de précision arbitraire							
Word8	Entier non signé de 8 bits							
Word16	Entier non signé de 16 bits							
Word32	Entier non signé de 32 bits							
Word64	Entier non signé de 64 bits							

Types numériques essentiels, d'après O'SULLIVAN et al. 2008

Types composites

Haskell connaît deux types composites : les **tableaux** et les **n-uplets**. Le type tableau se note `[a]`, un n-uplet se note `(a, b)`, `(a, b, c)`, *etc.*. Un n-uplet a au moins deux éléments, à l'exception de `()`, le n-uplet vide qui indique qu'une fonction ne renvoie pas de valeur.

Les tableaux sont typés : un tableau ne peut contenir des éléments que d'un type unique, pleinement paramétrisé. Par exemple, un tableau de type `[(Integer, Char)]`

(tableau de couples `Integer, Char`) ne peut pas contenir un autre type de n-uplet, par exemple (`Integer`, `String`).

2 Définition de types

Types personnalisés

On crée un type avec le mot-clé **data**. Au plus simple, un type réunit plusieurs variables membres :

```
1 ghci> data MonType = MonType String Integer
2 ghci> let a = MonType "abc" 123
```

(La première occurrence de MonType est le nom du type, la seconde est le nom du constructeur)

Synonymes de types

type crée un synonyme d'un type existant. Les synonymes et le type auquel ils renvoient sont interchangeables.

```
1 type ObjectId = Int16
```

Les synonymes créés avec **type** servent principalement à clarifier le sens des champs dans les types utilisateur.

```
1 type Authors = [String]
2 type Title = String
3 type ISBN = Int
4 type Year = Int
5 data Book2 = Authors Title Year ISBN
```

Synonymes « forts »

Maybe

Either

3 Contraintes de nommage

En Haskell, la casse du premier caractère d'un identifiant a une importance.

Type d'identifiant	Initiale haut de casse	Initiale bas de casse
Variable		■
Fonction		■
Type	■	
Constructeur	■	

4 Évaluation paresseuse

```
1 let a = [1..] -- a est la liste de l'ensemble des entiers positifs
2 let b = map ((^^) 2) a
```

L'évaluation paresseuse a un prix, qui est une plus grande consommation de mémoire : au lieu d'évaluer $2 + 2$, Haskell stocke un *thunk*, c'est à dire en gros un calcul différé. Mais sur les gros traitements récursifs, l'accumulation de thunks peut entrainer rapidement un débordement de mémoire. La commande `seq` force l'évaluation et permet d'éviter un débordement de mémoire.

seq

Lorem !

5 Fonctions et variables

Transparence référentielle : application de fonction = valeur

Syntaxe des prototypes de fonctions

[Le prototype restreint les actions possibles de la fonction]

Fonctions préfixes et infixes

6 Lambdas

7 Opérations sur des listes

8 Entrées/sorties et fonctions impures

type IO

9 Polymorphisme

Fonctions polymorphiques

Inférence de type

10 Classes de type

Les classes de type ne sont pas des classes au sens que ce terme possède en POO. Elles sont plus proches de ce qu'on nomme des interfaces : pour qu'un type appartienne à une classe, il faut implémenter les méthodes définies par la classe. Celle-ci peut fournir des implémentations de base,

11 Récursivité

Récursivité en queue

Folds

12 Structures de contrôle

`if ... then ... else`

Renvoie une valeur, par conséquent, le type de l'expression de `then` et de `else` doit être le même.

`case`

13 Application partielle et *currying*

Une fonction, quel que soit le nombre de paramètres avec lequel elle a été déclarée, ne prend qu'un seul paramètre et renvoie une autre fonction. Le type de `+`, par exemple, est : `Num a => Num a -> Num a -> Num a`, ce qui signifie que `+` prend un premier paramètre d'un type de type `Num`

14 Modules

Haskell dispose d'un mécanisme d'importation de modules.

Écrire un module

Un module a le même nom que le fichier .hs qui le contient, et ce nom commence par une majuscule ($\rightarrow 3$). La déclaration de module a la syntaxe suivante :

```
1  -- MyModule.hs
2  module Mod
3      (
4      x,
5      y,
6      z
7      ) where
8  -- code
```

Cette déclaration exporte les identifiants x, y et z du code qui la suit. On exporterait la totalité des noms en enlevant la parenthèse, et aucun en la laissant vide.

Importation de modules

	Importé
1 -- Commande	
2 import Mod	-- x, y, z, Mod.x, Mod.y, Mod.z
3 import Mod ()	-- (Nothing!)
4 import Mod (x,y)	-- x, y, Mod.x, Mod.y
5 import qualified Mod	-- Mod.x, Mod.y, Mod.z
6 import qualified Mod (x,y)	-- Mod.x, Mod.y
7 import Mod hiding (x,y)	-- z, Mod.z
8 import qualified Mod hiding (x,y)	-- Mod.z
9 import Mod as Foo	-- x, y, z, Foo.x, Foo.y, Foo.z
10 import Mod as Foo (x,y)	-- x, y, Foo.x, Foo.y
11 import qualified Mod as Foo	-- Foo.x, Foo.y, Foo.z
12 import qualified Mod as Foo (x,y)	-- Foo.x, Foo.y

D'après HUDAK et al. 2000

15 Composition

16 Monoïdes

17 Monades

Le wiki Haskell décrit les monades comme suit :

Monads in Haskell can be thought of as composable computation descriptions. The essence of monad is thus separation of composition timeline from the composed computation's execution timeline, as well as the ability of computation to implicitly carry extra data, as pertaining to the computation itself, in addition to its one (hence the name) output, that it will produce when run (or queried, or called upon). This lends monads to supplementing pure calculations with features like I/O, common environment or state, etc.

Références

- HUDAK, Paul, John PETERSON et Joseph FASEL (2000). *A Gentle Introduction to Haskell. Version 98*. URL : <https://www.haskell.org/tutorial/index.html>.
- O'SULLIVAN, Bryan, Don STEWART et John GOERZEN (2008). *Real World Haskell*. Anglais. O'Reilly. URL : <http://book.realworldhaskell.org/>.