Haskell

Voyage en terres inconnues - S01E03

Mikaël Bouillot - Meetup FP-MTP 2017

Haskell, c'est quoi?

- **un langage purement fonctionnel**
- **une évaluation "paresseuse" (lazy)**
- **typé à la Hindley-Milner:**
  - typage statique
  - polymorphisme paramétrique
  - inférence de types
  - étendu par des "classes de types"
- **un système d'entrées / sorties monadique**

Plan de la présentation

I. Histoire de Haskell
II. Introduction au langage
III. Apprendre Haskell

Ma rencontre avec Haskell

- Linux et le C
- Fusefilters: le C montre ses limites
- Haskell et la théorie des catégories
- Haskell en pratique

Histoire de Haskell

1977: Le prix Turing de John Backus

- Le créateur de FORTRAN (1957)
- "Can Programming be Liberated from the von Neumann Style?"
- FP: la programmation "function-level"
- Relance l'intérêt pour la programmation fonctionnelle en général

1977: Les précurseurs

- Lisp (1958)
- ML (1973)
- Scheme (1975)

1980: L'appel de la paresse

- **David Turner**
  - SASL (1976)
  - KRC (1982)
  - Miranda (1985)
- **Paul Hudak**
  - Alfl (1984)
- **Simon Peyton Jones**
  - Lazy ML (1984)
- **Philip Wadler**
  - Orwell (1984)
  - OL (1985)

1987: Le regroupement des troupes

Hudak, Wadler et Peyton Jones proposent de créer un nouveau langage fonctionnel pour unifier la communauté fragmentée

Ils demandent à Turner s'ils peuvent se baser sur Miranda, le langage a évaluation paresseuse le plus populaire à l'époque

Turner décline l'offre, souhaitant garder le contrôle de son langage

1988: Le comité Haskell

Les objectifs pour le langage:

- Adapté à la recherche et à l'enseignement, mais aussi au gros projets de développement
- Syntaxe et sémantique spécifiée formellement
- Librement utilisable (licence permissive)
- Utilisable pour de futurs travaux de recherche
- Basé sur des concepts déjà bien établis
- Réduisant la diversité superflue existant parmis les langage existants

1988: Le comité Haskell

Lors de la première réunion, le projet est également baptisé: "Haskell", en l'honneur du mathématicien Haskell Curry

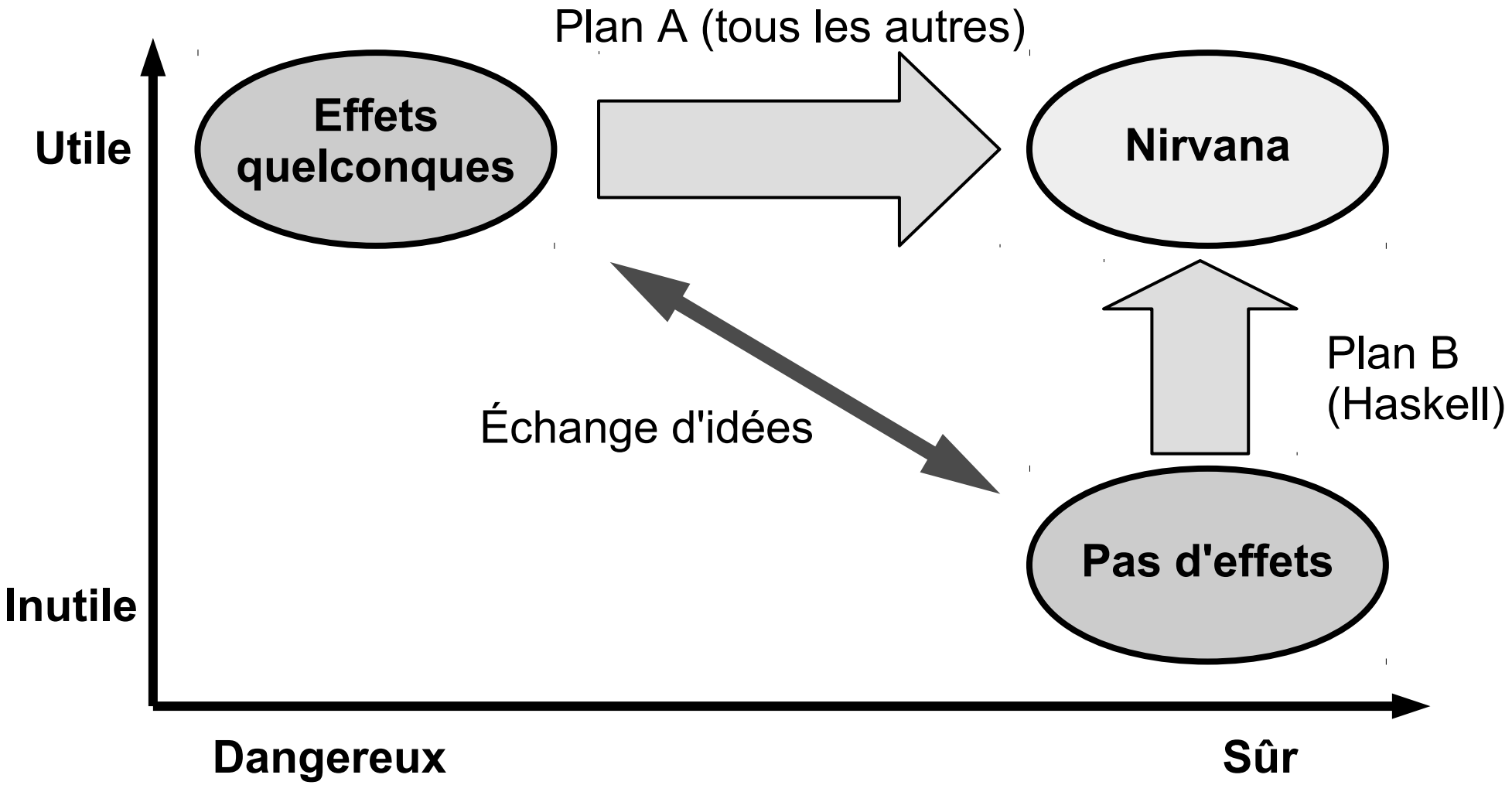
1992: Les entrées / sorties monadiques

- Haskell 1.0: streams ou continuations
- Eugenio Moggi (1989) introduit les monades pour décrire les effets de bord de manière théorique
- Wadler (1992) propose d'implémenter les monades en Haskell afin de manipuler les actions à effets de bord
- La monade IO devient la méthode standard pour effectuer des entrées / sorties en Haskell
- La "do-notation" est introduite, immitant la syntaxe des langages impératifs

1999: Le rapport Haskell 98

En 1999, le rapport final du comité Haskell est publié sous le nom de "Haskell 98". Le comité est alors dissout afin de laisser le langage évoluer plus librement.

Présent et futur



Introduction au langage

Termes, fonctions et évaluation

Définir une fonction: lambda abstractions et équations

```
\x -> x*x

sqr = \x -> x*x

sqr x = x*x
```

Termes, fonctions et évaluation

Appliquer une fonction à un argument: juxtaposition

```
> (\x -> x*x) 4
16

> sqr x = x*x
> sqr 4
16
```

Termes, fonctions et évaluation

Fonctions de plusieurs arguments: la curryfication

```
> add 2 3
5

> (add 2) 3
5

> (\x -> add 2 x) 3
5
```

Termes, fonctions et évaluation

Fonctions préfixes et opérateurs infixes

```
> 2+3
5

> (+) 2 3
5

> (+2) 3
5
```

Termes, fonctions et évaluation

Les listes: construction et extraction

```
> (:) 1 [2,3,4]
[1,2,3,4]

> 1 : 2 : 3 : 4 : []
[1,2,3,4]

> head [1,2,3,4]
1

> tail [1,2,3,4]
[2,3,4]
```

Termes, fonctions et évaluation

Évaluation paresseuse et listes infinies

```
> ones = 1 : ones
> twos = map (*2) ones
> take 3 twos
[2,2,2]

> nat = 0 : map (+1) nat
> take 10 nat
[0,1,2,3,4,5,6,7,8,9]
```

Les types

Indiquer le type d'un terme: les signatures

```
           2 :: Int
        3.14 :: Float
        True :: Bool

     [1,2,3] :: [Int]
[True, False] :: [Bool]

        sqrt :: Float -> Float
        even :: Int -> Bool
```

(note: on simplifie pour l'instant certain types)

Les types

Type des fonctions curryfiées

```
add :: Int -> Int -> Int
add :: Int -> (Int -> Int)

add 2 :: Int -> Int

add 2 3 :: Int
```

Les types

Types polymorphiques et variables de types

```
(:) :: a -> [a] -> [a]

head :: [a] -> a
tail :: [a] -> [a]
length :: [a] -> Int

map :: (a -> b) -> [a] -> [b]
```

Types de données algébriques

Définir de nouveaux types: les data-types (ADT)

```
data ColorSum = Yellow | Pink | Purple | Brown

data ColorProd = Color Float Float Float

data Result = Error String | Success Int
```

Types de données algébriques

Types polymorphiques et types récursifs

```
data Maybe a = Nothing | Just a

data Either a b = Left a | Right b

data List = Nil | Cons Int List
```

```
data List a = Nil | Cons a (List a)

data [a] = [] | a : [a]     <-- not real Haskell
```

Types de données algébriques

Gravir l'abstraction des types: les higher-kinded types

```
data Wat a = Wt (a Int)

Wt [1,2,3]  :: Wat []
Wt (Just 2) :: Wat Maybe
```

Les entrées / sorties

Représenter des actions à effets de bord: le type "IO"

```
getLine :: IO String
putStr  :: String -> IO ()

(>>)    :: IO a -> IO b -> IO b
(>>=)   :: IO a -> (a -> IO b) -> IO b
return :: a -> IO a

getLine >>= putStr :: IO ()
```

Les entrées / sorties

Simplifier la syntaxe: la do-notation

```
action1 >>= (\a ->
action2 >>= (\b ->
action3 >>= (\c ->
return (f a b c))))
```

```
do a <- action1
   b <- action2
   c <- action3
   return (f a b c)
```

```
main = do
  putStrLn "What's your name?"
  name <- getLine
  putStrLn ("Hello, " ++ name)
```

Les classes de types

Restreindre le champ des possibles: le polymorphisme ad-hoc

```
(+)  :: Num a => a -> a -> a
(==) :: Eq a  => a -> a -> Bool
(>)  :: Ord a => a -> a -> Bool

class Num a where
  (+) :: a -> a -> a
  (*) :: a -> a -> a

instance Num Int where
  (+) x y = ...
  (*) x y = ...
```

Foncteurs et monades

Généraliser la fonction "map": les foncteurs

```
map :: (a -> b) -> [a] -> [b]

class Functor f where
  fmap :: (a -> b) -> f a -> f b

data List a = Nil | Cons a (List a)

instance Functor List where
  fmap f Nil = Nil
  fmap f (Cons x xs) = Cons (f x) (fmap f xs)
```

Foncteurs et monades

Généraliser "bind" et "return": les monades

```
class Monad m where
  (>>=)  :: m a -> (a -> m b) -> m b
  (>>)   :: m a -> m b -> m b
  return :: a -> m a

instance Monad IO where
  ...
```

Apprendre Haskell

Apprendre Haskell, pourquoi?

- Découvrir les systèmes de types HM + extensions
- Prototyper des projets:
  - Langage de très haut niveau
  - Type system expressif et safe
  - Évaluation paresseuse
  - Environnement interactif
- Comprendre les types abstraits
- Découvrir la théorie des catégories?

Apprendre Haskell, comment?

- "A Gentle Introduction to Haskell" - P. Hudak
- "A History of Haskell" - S. Peyton Jones (2007)
  - Paper (~50 pages)
  - Présentation sur YouTube (1 heure)
- "Monads for Functional Programming" - P. Wadler (1995)

Les projets en Haskell

- Les compilateurs:
  - GHC (Haskell)
  - Pugs (Perl 6)
  - Elm
- Darcs (gestionnaire de version distribué)
- Servant (serveur HTTP)
- hledger (comptabilité)

Haskell dans l'industrie

- Vente Privée
- Clever Cloud
- FretLink
- Inria

Voir aussi: https://wiki.haskell.org/Haskell_in_industry