

# Heterogeneous-Arrow

## Modéliser le rythme du Data-Flow

Gautier DI FOLCO, Master II en Recherche Réseaux Télécoms et Services, Université Lyon 1.

Laboratoire CITI, sous la direction de Stéfane Frénot.

Lyon, 23/06/14



# Big data

- Afflux de données très important
- Volume de stockage important
- Nécessité de traitement haute fréquence des données
- Nécessité d'accumuler indéfiniment les données



# Augmentation des performances

- Des infrastructures matérielles
  - Datacenter
  - Économie
- Des infrastructures logicielles
  - Hadoop (batch)
  - Storm (flux)



## Gestion du Data-flow

- Début inaccessible et sans fin
- Gigue plus importante que le vitesse
- Middleware + Infrastructure

=> Approche de la gestion du flux au niveau du langage de programmation initial



# Définir un langage de programmation

- Formaliser les flux
- Formaliser des algorithmes de traitements
- Offrant des garanties de preuve sur les délais de bout en bout



# Un langage de programmation

- 1974 : Réseaux de Kahn
  - Divise le programme en entités
  - Communication par **channels**, des listes FIFO
  - Aucune implantation direct



## Formaliser les flux

- 1976 : Lucid
  - Se base sur des suites, premier élément et méthode pour calculer les suivants
  - Le flux est tronqué pour obtenir un résultat
- 1984 : Esterel
  - Le programme n'a qu'une entrée et qu'une sortie
  - Nous sommes limité à la transformation d'un flux



## Formaliser des algorithmes de traitements

- 1994 : TBAG
  - Modélise les variations temporelles des relations
  - Ne gère pas les traitements
- 1995 : Mediaflow
  - Combinaison visuelle des flux
  - Pas de notion de traitements





## Offrant les meilleurs garanties de preuve sur les délais de bout en bout

- 1997 : Fran
  - Fondateur du Functional Reactive Programming
  - Modélise la composition de traitements
  - Aucune gestion des délais
- 2002 : RT-FRP
  - Approche du temps réel par calcul des coûts de traitements
  - Aucune garantie globale



	Kahn	Lucid	Esterel	TBAG	Mediaflow	Fran	RT-FRP
Langage	X	V	V	X	X	V	V
Structure	V	X	X	V	V	V	V
Algorithmes	X	X	X	X	X	V	V
Preuves	X	X	V	X	X	X	X



# Langage de programmation de flux

- Formel
  - Typage fort
    - Haskell (Théorie des catégories)
    - Type  $\Rightarrow$  Flux (**Arrow**)



## Composition de 2 Arrow

On compose deux valeurs de même type :

$$\frac{A \boxed{\text{f}}}{\text{f}} . \frac{B \boxed{\text{g}}}{\text{g}} = \frac{A \boxed{\text{h}}}{\text{h}}$$

On compose deux valeurs de types différents :

$$\frac{A \boxed{\text{f}}}{\text{f}} . \frac{B \boxed{\text{g}}}{\text{g}} = \frac{A \boxed{\text{?}}}{\text{?}}$$



## Mon travail

- Partir des Arrows
- Résoudre la composition
  - Rythme : Débit de consommation et de production des données
  - Nature : Besoin d'un état ou non



## Notes

- Utilisation des **Category** qui sont une généralisation des **Arrow**
- Décrit la notion de composition

$$\begin{array}{c} \text{comme} \\ \underline{A} \boxed{f} . \underline{B} \boxed{g} = \underline{A} \boxed{h} \\ \text{ou} \\ \underline{A} \frown f . \underline{B} \frown g = \underline{A} \frown h \end{array}$$



# Notes

- Utilisation des **Arrow**
- Décrit les dérivation automatiques

Non-composable :

$$\frac{A}{\text{f}} \cdot \frac{B}{\text{c}}$$

Composable :

$$\frac{A}{\text{f}} \cdot \frac{B}{\text{c}} = \frac{A}{\text{h}}$$



## Mise en oeuvre N°1

- Utilisation des **Kleisli Arrow**
- Stock l'état d'un traitement

Non-composable :



Revient à faire :





## Mise en oeuvre N°2

- Utilisation des **Heterogeneous-Arrows**
- Définit les compositions une à une
- Comme les transitions d'une machine à états

Non-composable :

$$\frac{A}{f} . \frac{B}{g} = \frac{A}{h}$$



# Conclusion

- Un début : généralisation basique du concept de composition
- Généralisable :
  - Rapide + Lent = Lent
  - Duplication/parallélisation des opérations n'ayant pas d'état



# Questions

Merci



```
class Category cat where
  id :: cat a a
  (.) :: cat b c -> cat a b -> cat a c

instance Category (=>) where
-- Version prefix : id :: (=>) a a
-- Version infix  : id :: a -> a
  id x = x

-- Version prefix : (.) :: (=>) b c -> (=>) a b -> (=>) a c
-- Version infix  : (.) :: (b -> c) -> (a -> b) -> (a -> c)
  f . g = \x -> f (g x)

class Category a => Arrow a where
  arr :: (b -> c) -> a b c
  first :: a b c -> a (b, d) (c, d)
  second :: a b c -> a (d, b) (d, c)
  (***) :: a b c -> a b' c' -> a (b, b') (c, c')
  (&&&) :: a b c -> a b c' -> a b (c, c')
```



```
class HeterogeneousArrow x y z | x y -> z where
  (>*>) :: x a b -> y b c -> z a c

instance (Category x) => HeterogeneousArrow x x x where
  f >*> g = g . f
```

