

# Projet de recherche

## Résumé de l'article **Handling Algebraic effects**

Jordan Ischard

Université d'Orléans

2020-2021

# Sommaire

## Introduction

Problème de l'impératif dans un langage fonctionnel  
Proposition pour gérer les effets

Gestion des effets dans l'article

Validé des gestionnaires

Confrontation

Conclusion

# Sommaire

## Introduction

Problème de l'impératif dans un langage fonctionnel

Proposition pour gérer les effets

## Gestion des effets dans l'article

## Validé des gestionnaires

## Confrontation

## Conclusion

# Ajout de principes impératifs dans un langage fonctionnel

But

Ajouter des fonctionnalités en plus dans les langages fonctionnels

# Ajout de principes impératifs dans un langage fonctionnel

## But

Ajouter des fonctionnalités en plus dans les langages fonctionnels

## Problème

Ajoute des effets de bords que les langages fonctionnels purs non pas.

# Ajout de principes impératifs dans un langage fonctionnel

## But

Ajouter des fonctionnalités en plus dans les langages fonctionnels

## Problème

Ajoute des effets de bords que les langages fonctionnels purs non pas.

## Exemple

On peut avoir des effets de bords pour les appels mémoires ou encore les entrées/sorties.

# Ajout de principes impératifs dans un langage fonctionnel

## But

Ajouter des fonctionnalités en plus dans les langages fonctionnels

## Problème

Ajoute des effets de bords que les langages fonctionnels purs non pas.

## Exemple

On peut avoir des effets de bords pour les appels mémoires ou encore les entrées/sorties.

## Idée

Créer une structure qui va gérer ces effets.

# Sommaire

## Introduction

Problème de l'impératif dans un langage fonctionnel

Proposition pour gérer les effets

## Gestion des effets dans l'article

## Validé des gestionnaires

## Confrontation

## Conclusion



# Réponses déjà proposées

## Les Monades

Eugenio Moggi a proposé le principe de **Monade** pour gérer les effets.

# Réponses déjà proposées

## Les Monades

Eugenio Moggi a proposé le principe de **Monade** pour gérer les effets.

## Les Monades en Haskell

```
1 instance Monad Maybe where
2     return x = Just x
3
4     Nothing >>= f = Nothing
5     Just x >>= f  = f x
6
7     fail _ = Nothing
8
```

# Réponses déjà proposées

## Les Monades

Eugenio Moggi a proposé le principe de **Monade** pour gérer les effets.

## Les Monades en Haskell

```
1 instance Monad Maybe where
2     return x = Just x
3
4     Nothing >>= f = Nothing
5     Just x >>= f = f x
6
7     fail _ = Nothing
8
```

## Les théories d'équations

Plotkin et Power ont proposé des opérations comme source des effets et une **théorie d'équation** pour décrire leurs propriétés.

# Sommaire

Introduction

Gestion des effets dans l'article

Intuition et principe

Mise en place dans une syntaxe

Exemple

Validé des questionnaires

Confrontation

Conclusion

# Sommaire

## Introduction

## Gestion des effets dans l'article

### Intuition et principe

Mise en place dans une syntaxe

Exemple

## Validé des questionnaires

## Confrontation

## Conclusion

# Comment gérer les effets ?

## Intuition

Reprendre le principe de Plotkin et Power et l'associer aux travaux de Benton et Kennedy sur les gestionnaires.

# Comment gérer les effets ?

## Intuition

Reprendre le principe de Plotkin et Power et l'associer aux travaux de Benton et Kennedy sur les gestionnaires.

## Principe

Les effets vont avoir pour source des **opérations** et vont être géré par un **tableaux associatifs (map)** contenu dans un gestionnaire.

# Sommaire

## Introduction

## Gestion des effets dans l'article

Intuition et principe

Mise en place dans une syntaxe

Exemple

## Validé des questionnaires

## Confrontation

## Conclusion



# Qu'ajoute-on pour mettre en place la gestion des effets ?

## Syntaxe ajouté

On part de la syntaxe proposé par Levy dans son article sur *call-by-push-value*.

*source de l'effet* :  $\mathbf{op}_V(x : \beta.M')$

*structure de gestion* :  $M$  **handled with**  $H$  *to*  $x : A.N$

*gestionnaire* :  $\{\mathbf{op}_{z:\alpha}(k : \beta \rightarrow \underline{C}) \mapsto M_{\mathbf{op}}\}_{\mathbf{op}:\alpha \rightarrow \beta}$

# Qu'ajoute-on pour mettre en place la gestion des effets ?

## Syntaxe ajouté

On part de la syntaxe proposé par Levy dans son article sur *call-by-push-value*.

*source de l'effet* :  $\mathbf{op}_V(x : \beta.M')$

*structure de gestion* :  $M$  **handled with**  $H$  *to*  $x : A.N$

*gestionnaire* :  $\{\mathbf{op}_{z:\alpha}(k : \beta \rightarrow \underline{C}) \mapsto M_{\mathbf{op}}\}_{\mathbf{op}:\alpha \rightarrow \beta}$

## Fonctionnement

On reprend la structure de gestion ci-dessus avec  $\mathbf{op}_V(y.M') \in M$  et  $\{\mathbf{op}_z(k) \mapsto M_{\mathbf{op}}\} \in H$ .

$M_{\mathbf{op}}[V/z, M'[W/y]$  **handled with**  $H$  **to**  $x : A.N/k(W)]$

# Sommaire

## Introduction

## Gestion des effets dans l'article

Intuition et principe

Mise en place dans une syntaxe

**Exemple**

## Validé des questionnaires

## Confrontation

## Conclusion

# dérogation à la lecture seule

## Valeur Temporaire sans modification dans la mémoire

Le gestionnaire définit pour l'effet est le suivant :

$$H_{temporary} = \{\mathbf{get}_{l:\mathbf{loc}}(k : \mathbf{nat} \rightarrow \underline{C}) \mapsto k(n)\}$$

Prenons l'expression suivante :

```
let  $n : \mathbf{nat}$  be 20 in  
get $l$ ( $x : \mathbf{nat}.$ get $l$ ( $y : \mathbf{nat}.$ return  $x + y$ ))  
handled with  $H_{temporary}$  to  $z : A.$ return  $z + 2$ 
```

# dérogation à la lecture seule

## Valeur Temporaire sans modification dans la mémoire

Le gestionnaire définit pour l'effet est le suivant :

$$H_{temporary} = \{\mathbf{get}_{l:\text{loc}}(k : \mathbf{nat} \rightarrow \underline{C}) \mapsto k(\textcolor{red}{n})\}$$

Prenons l'expression suivante :

**let  $n : \mathbf{nat}$  be 20 in**

**get<sub>l</sub>( $x : \mathbf{nat}$ .get<sub>l</sub>( $y : \mathbf{nat}$ .return  $x + y$ ))**

**handled with  $H_{temporary}$  to  $z : A$ .return  $z + 2$**

# dérogation à la lecture seule

## Valeur Temporaire sans modification dans la mémoire

Le gestionnaire définit pour l'effet est le suivant :

$$H_{\text{temporary}} = \{\text{get}_{y:\text{loc}}(k : \text{nat} \rightarrow \underline{C}) \mapsto k(20)\}$$

Prenons l'expression suivante :

**get<sub>l</sub>**(**x** : **nat**.**get<sub>l</sub>**(**y** : **nat**.**return** **x** + **y**))  
**handled with** *H<sub>temporary</sub>* **to** **z** : **A**.**return** **z** + 2

# dérogation à la lecture seule

## Valeur Temporaire sans modification dans la mémoire

Le gestionnaire définit pour l'effet est le suivant :

$$H_{\text{temporary}} = \{\text{get}_{y:\text{loc}}(k : \text{nat} \rightarrow \underline{C}) \mapsto k(20)\}$$

Prenons l'expression suivante :

**get<sub>l</sub>**(*y* : **nat**.**return** 20 + *y*)

**handled with** *H<sub>temporary</sub>* **to** *z* : *A*.**return** *z* + 2

# dérogation à la lecture seule

## Valeur Temporaire sans modification dans la mémoire

Le gestionnaire définit pour l'effet est le suivant :

$$H_{temporary} = \{\mathbf{get}_{y:\mathbf{loc}}(k : \mathbf{nat} \rightarrow \underline{C}) \mapsto k(20)\}$$

Prenons l'expression suivante :

**return** 20 + 20

**handled with**  $H_{temporary}$  **to**  $z : A.$ **return**  $z + 2$



# dérogation à la lecture seule

## Valeur Temporaire sans modification dans la mémoire

Le gestionnaire définit pour l'effet est le suivant :

$$H_{temporary} = \{\mathbf{get}_{y:\mathbf{loc}}(k : \mathbf{nat} \rightarrow \underline{C}) \mapsto k(20)\}$$

Prenons l'expression suivante :

**return** 40 + 2

# dérogation à la lecture seule

## Valeur Temporaire sans modification dans la mémoire

Le gestionnaire définit pour l'effet est le suivant :

$$H_{temporary} = \{\mathbf{get}_{y:\mathbf{loc}}(k : \mathbf{nat} \rightarrow \underline{C}) \mapsto k(20)\}$$

Prenons l'expression suivante :

**return** 40 + 2

La réponse

C'est 42 !

# Sommaire

Introduction

Gestion des effets dans l'article

**Validé des questionnaires**

Gestionnaire générique indécidable

Sous-ensemble décidable

Confrontation

Conclusion

# Sommaire

Introduction

Gestion des effets dans l'article

Validé des questionnaires

Gestionnaire générique indécidable

Sous-ensemble décidable

Confrontation

Conclusion

# Les gestionnaires en général sont indécidable

## Limite des gestionnaires

En général la validité des gestionnaires est **indécidable**.

# Les gestionnaires en général sont indécidable

## Limite des gestionnaires

En général la validité des gestionnaires est **indécidable**.

## Comment palier à se problème

Deux approches sont possibles :

1. Obliger l'utilisateur a utiliser des gestionnaires que l'on sait valide (donc prédéfinis dans le langage) : **la responsabilité est au créateur du langage**
2. Laisser l'utilisateur faire comme il le souhaite tout en lui laissant la possibiliter de faire des gestionnaires invalide : **la responsabilité est à l'utilisateur**.

# Sommaire

Introduction

Gestion des effets dans l'article

Validé des questionnaires

Gestionnaire générique indécidable

Sous-ensemble décidable

Confrontation

Conclusion

# Gestionnaire décidable sous certaines conditions

TODO : principe de gestionnaire simple, de famille uniformément simple etc.



# Sommaire

Introduction

Gestion des effets dans l'article

Validé des questionnaires

**Confrontation**

Les divergences

Exemple de conversion entre langage

Difficulté d'implémentation

Conclusion

# Sommaire

Introduction

Gestion des effets dans l'article

Validé des questionnaires

Confrontation

Les divergences

Exemple de conversion entre langage

Difficulté d'implémentation

Conclusion

# Motivation et fonctionnement divergent

Langage de l'article	Langage <b>erpl</b>
<ul style="list-style-type: none"><li>○ Créer pour voir l'aspect mathématique de la gestion d'effets algébriques</li></ul>	<ul style="list-style-type: none"><li>○ Créer pour avoir une version allégé du langage Ocaml</li></ul>

# Motivation et fonctionnement divergent

Langage de l'article	Langage <b>erpl</b>
<ul style="list-style-type: none"><li>○ Créer pour voir l'aspect mathématique de la gestion d'effets algébriques</li><li>○ La source d'un effet est une <b>opération</b></li></ul>	<ul style="list-style-type: none"><li>○ Créer pour avoir une version allégé du langage Ocaml</li><li>○ La source d'un effet est un <b>type</b></li></ul>

# Motivation et fonctionnement divergent

Langage de l'article	Langage <b>erpl</b>
<ul style="list-style-type: none"><li>○ Créer pour voir l'aspect mathématique de la gestion d'effets algébriques</li><li>○ La source d'un effet est une <b>opération</b></li><li>○ Activation <b>implicite</b> de l'effet</li></ul>	<ul style="list-style-type: none"><li>○ Créer pour avoir une version allégé du langage Ocaml</li><li>○ La source d'un effet est un <b>type</b></li><li>○ Activation <b>explicite</b> de l'effet</li></ul>

# Motivation et fonctionnement divergent

Langage de l'article	Langage <b>erpl</b>
<ul style="list-style-type: none"><li>○ Créer pour voir l'aspect mathématique de la gestion d'effets algébriques</li><li>○ La source d'un effet est une <b>opération</b></li><li>○ Activation <b>implicite</b> de l'effet</li><li>○ Gestionnaire global implicite car l'activation est implicite</li></ul>	<ul style="list-style-type: none"><li>○ Créer pour avoir une version allégé du langage Ocaml</li><li>○ La source d'un effet est un <b>type</b></li><li>○ Activation <b>explicite</b> de l'effet</li><li>○ Pas de gestionnaire global, tout activation sans gestionnaire provoque une erreur</li></ul>

Table: Comparaison entre les deux langages

# Sommaire

Introduction

Gestion des effets dans l'article

Validé des questionnaires

Confrontation

Les divergences

Exemple de conversion entre langage

Difficulté d'implémentation

Conclusion

# Passage du langage de l'article vers **erpl**

## Reprise de l'exemple précédent

**let**  $n : \text{nat}$  **be** 20 **in**

**get** <sub>$I$</sub> ( $x : \text{nat}$ .**get** <sub>$I$</sub> ( $y : \text{nat}$ .**return**  $x + y$ ))

**handled with**  $\{\text{get}_{I:\text{loc}}(k : \text{nat} \rightarrow \underline{C}) \mapsto k(n)\}$  **to**  $z : A$ .**return**  $z + 2$



# Passage du langage de l'article vers **erpl**

## Reprise de l'exemple précédent

**let**  $n : \text{nat}$  **be** 20 **in**

**get** <sub>$l$</sub> ( $x : \text{nat}$ .**get** <sub>$l$</sub> ( $y : \text{nat}$ .**return**  $x + y$ ))

**handled with** {**get** <sub>$l$</sub> .loc( $k : \text{nat} \rightarrow \underline{C}$ )  $\mapsto k(n)$ } **to**  $z : A$ .**return**  $z + 2$

## Reprise de l'exemple converti en **erpl**

```
1 type memory = Get of 'a ref -> 'a
2               | Set of 'a ref * 'a -> unit;;
3
4 let get = Get(fun x -> !x) and n = 20 in
5     handle ((fun x -> fun y -> x + y + 2)
6             (perform (get,l))) (perform (get,l))
7     with (Get f,x),k -> k n ;;
8
```

# Sommaire

Introduction

Gestion des effets dans l'article

Validé des questionnaires

Confrontation

Les divergences

Exemple de conversion entre langage

Difficulté d'implémentation

Conclusion

# Une histoire d'appels systèmes

## Comment ça fonctionne ?

En prenant, le principe d'opération comme source d'effet on implique une gestion des opérations particuliers.

Quand on a pas de gestionnaire, les opérations se *gèrent elles-même*.

# Une histoire d'appels systèmes

## Comment ça fonctionne ?

En prenant, le principe d'opération comme source d'effet on implique une gestion des opérations particuliers.

Quand on a pas de gestionnaire, les opérations se *gèrent elles-même*.

## Idée

Créer un gestionnaire globale implicite ajouté à la compilation.

# Une histoire d'appels systèmes

## Comment ça fonctionne ?

En prenant, le principe d'opération comme source d'effet on implique une gestion des opérations particuliers.

Quand on a pas de gestionnaire, les opérations se *gèrent elles-même*.

## Idée

Créer un gestionnaire globale implicite ajouté à la compilation.

## Problèmes d'appels systèmes

Cela veut dire que notre gestionnaire globale doit être capable de faire des appels systèmes et donc que le langage en soit capable.

**Pas facilement intégrable !**

# Sommaire

Introduction

Gestion des effets dans l'article

Validé des questionnaires

Confrontation

Conclusion

# Que peut-on conclure ?

TODO : No se