Résumé de l'article **Handling algebraic effects**

Jordan Ischard[1]

[1] *Université d'Orléans*

Table des matières

Introduction

Le pionnier dans la gestion des effets algébriques est **Eugenio Moggi**. Il proposa une représentation uniforme de calcul d'effets par les **Monades** [1].

Exemple 1. On prend un ensemble de catégorie A. Un calcul qui retourne une valeur $a \in A$ est modélisé par $ta \in TA$ pour une monade T correspondante.

Les effets et leurs gestions sont utiles pour représenter les exceptions, les états mémoires, le non-déterminisme, les I/O, etc. **Moggi** ne fut pas le seul à proposer une gestion des effets, en effet **Plotkin et Power** proposerons plus tard une représentation basée sur un ensemble d'opérations qui représente la sources des effets et une théorie d'équations, pour ces dites opérations, qui décrit leurs propriétés.

L'intuition derrière est que chaque calcul retourne soit une **valeur**, soit effectue une **opération** avec un retour déterminant la continuation. Les arguments de cette opération représente donc les potentiels continuations.

Exemple 2. On prend une opération binaire de choix **choose**. C'est une opération qui choisit de manière non déterministe un booléen à retourner :

$$\textbf{choose(return true,return false)}$$

On a deux possibilités : soit on continue avec le calcul donné par le première argument soit on continue avec le calcul donné par le second.

Les deux représentations permettent de géré les effets et on peut passer d'une représentation à l'autre. Les effets représentables par une monade et une représentation équationnelle sont appelés **effets Algébriques**. Tous les effets cités ci-dessus sont algébriques.

La vision algébrique des effets a permis de les combiner et de raisonner dessus. Cependant, le gestionnaire d'exception apporte un challenge.

Soit la monade $\_ + \textbf{exc}$ pour l'ensemble des exceptions **exc**. Cette monade est représentée par une opération $\textbf{raise}_e()$ pour chaque $e \in \textbf{exc}$ et aucune équations. $\textbf{raise}_e()$ ne prend par d'arguments et il n'y a pas de continuation directement après une levée d'exception. Maintenant la monade définit, comment créer le gestionnaire d'exception ? L'approche la plus simple est la suivante:

$$\textbf{handle}_e(M, N)$$

M s'effectue à moins qu'une exception e soit levée, dans ce cas on effectue N. Cette construction binaire manque d'une certaine propriété caractérisant les opérations spécifiés dans les équations qui les décrivent.

Exemple 3. Ce qui est décrit ci-dessus correspond à l'opération de commutation avec le contexte d'évaluation $\varepsilon[\_]$.
On aimerait garder les équations suivantes :

$$\varepsilon[\textbf{choose}(M, N)] = \textbf{choose}(\varepsilon[M], \varepsilon[N]) \qquad \varepsilon[\textbf{raise}_e()] = \textbf{raise}_e()$$

mais pas :

$$\varepsilon[\textbf{handle}(M, N)] = \textbf{handle}(\varepsilon[M], \varepsilon[N])$$

Or les propriétés naturelles sont communes à toutes les opérations spécifiées par des équations, donc la solution proposée n'est pas suffisante.

Dans le papier on donne une explication algébrique des gestionnaires d'exceptions. L'idée principale est que :

1. les gestionnaires correspondent à des modèles de théorie équationnelle;
2. la sémantique des gestionnaires est donné à l'aide d'homomorphismes uniques qui ciblent de tels modèles et est induit par la propriété universelle du modèle libre.

La construction habituelle des gestionnaires d'exceptions correspond à l'application d'un homomorphisme unique qui préserve la valeur de retour. Ici, on adopte une vision plus générale suggéré par Benton et Kennedy. Dans leur approche, la valeur de retour est passé à la continuation définit par l'utilisateur. Cette idée se généralise à tous les effets algébrique, permettant à un nouveau concept de programmation de gérer les effets algébriques. Conceptuellement, les opérations algébriques et les gestionnaires d'effets sont duals : le premier peut être nommé *constructeur d'effets* car il crée l'effet et le second peut être nommé *destructeur d'effets* car il produit un calcul en lien avec l'effet créé.

Le but est de résumer l'article en gardant le maximum d'informations tout en omettant les parties les plus compliqués ou en les simplifiant. On va suivre les sections présentent dans l'article de base. Les 3 premières sections seront résumé tout en restant assez authentique au matériau d'origine. La section 4 utilise la théorie des catégorie et parle de l'interprétation de la théorie des effets. Cette partie étant hors de ma portée je n'en parlerait pas. Après cela, les sections 5 et 6 parlent de la décidabilité et du raisonnement des gestionnaires. On restera assez proche de ce qui est dit dans l'article. Et enfin, la section 7 sera à peine évoqué car elle traite la récursion qui est certes intéressante mais pas centrale dans l'article.

1 Gestionnaire des exceptions

On commence l'étude sur la gestionnaire des exceptions, d'une part pour comprendre le principe et d'autre part c'est un effet simple. La section est écrite de manière informelle.

On considère un ensemble fini d'exceptions **exc**, un second ensemble A et une monade d'exception TA telle que $TA \overset{\textbf{def}}{=} A + \textbf{exc}$. Un calcul retournant une valeur $a \in A$ est modélisé par un élément $ta \in TA$. Cette monade a pour unité $\eta_A : A \to A + \textbf{exc}$, et le calcul $return(V)$ est interprété par $\eta_A(V)$ tandis que $\textbf{raise}_e()$ est interprété par $in_2(e)$.

1.1 Construction des gestionnaires étendus

Benton et Kennedy ont généralisé la construction des gestionnaires avec la forme suivante :

$$M \textbf{ handled with}\{\textbf{raise}_e() \mapsto N_e\}_{e \in \textbf{exc}} \textbf{ to } x : A.N(x)$$

avec $\{...\}_{e \in \textbf{exc}}$ représentant l'ensemble des calculs, un pour chaque $e \in \textbf{exc}$. Dans cette construction, on effectue le calcul de $M \in A + \textbf{exc}$. Si on lève une exception $e \in \textbf{exc}$ alors on effectue un calcul N_e qui retourne un élément de B (où B peut différer de A). Sachant que N_e peut lui même lever une exception alors $N_e \in B + \textbf{exc}$. Les valeurs retournées par le gestionnaire sont passées dans une continuation défini par l'utilisateur $N : A \to B + \textbf{exc}$. La construction satisfait 2 équations :

$$\textbf{return } V \textbf{ handled with}\{\textbf{raise}_e() \mapsto N_e\}_{e \in \textbf{exc}} \textbf{ to } x : A.N(x) = N(V)$$
$$\textbf{raise}_{e'}() \textbf{ handled with}\{\textbf{raise}_e() \mapsto N_e\}_{e \in \textbf{exc}} \textbf{ to } x : A.N(x) = N_{e'}$$

Comme discuté dans [2], Cette construction permet d'écrire de façon simple un idiome de programmation qui était lourd à mettre en place.

Algébriquement le calcul N_e donne un nouveau modèle $\mathcal{M}$ pour la théorie des exceptions. Le porteur de ce modèle est $B + \textbf{exc}$. Pour chaque $e \in \textbf{exc}$, $\textbf{raise}_e()$ est interprété par N_e. On peut voir d'après les deux équations ci-dessus que :

$$h(M) \overset{def}{=} M \textbf{ handled with}\{\textbf{raise}_e() \mapsto N_e\}_{e \in \textbf{exc}} \textbf{ to } x : A.N(x)$$

où $h : A + \textbf{exc} \to \mathcal{M}$ est l'homomorphisme unique qui étend N, i.e, tel que l'on a le diagramme de commutation suivant :

$$\begin{tikzcd} A \arrow[rr, "\eta_A"] \arrow[dr, "N"'] & & A + \textbf{exc} \arrow[dl, "h"] \\ & \mathcal{M} & \end{tikzcd}$$

Notons que tous les homomorphismes d'un modèle libre vers un modèle pour un porteur donné sont obtenus de cette manière. Donc la construction proposé par Benton et Kennedy est le plus général possible d'un point de vue algébrique.

1.2 Gestion des effets algébriques arbitraires

Avec la construction étendue de Benton et Kennedy, on voit maintenant comment créer les gestionnaires pour les autres effets algébriques. Un **modèle d'une théorie d'équations** est une interprétation, i.e, un ensemble de tableaux associatifs, un pour chaque opération, qui satisfont les équations. Les gestionnaires permettent de telles interprétations. Comme avant, les calculs sont interprétés par le modèle libre et la construction des gestionnaires sont interprétés par l'homomorphisme induit. Avant, les exceptions étaient remplacés par un calcul du gestionnaire; maintenant les **opérations** sont remplacées par les tableaux associatifs du gestionnaire.

Il est important de noter que toute interprétation ne donne pas forcément un modèle de la théorie des équations et que donc tous les gestionnaires ne sont pas forcément correctes. Plus que ça, la validité d'un gestionnaire peut être indéfini. Pour contourner le problème, il y a deux écoles. Soit on contraint l'utilisateur à des gestionnaires prédéfinies que l'on sait correcte; soit on laisse une totale liberté à l'utilisateur mais c'est à lui qui est responsable de la validité de ses gestionnaires. On adoptera la seconde dans ce papier.

2 Syntaxe

2.1 Signatures

Les types de signature α, β sont définis par:

$$\alpha, \beta := \mathbf{b} \mid \mathbf{1} \mid \alpha \times \beta \mid \sum_{l \in L} \alpha_l$$

où $\mathbf{b}$ représente l'ensemble des types de base, L représente un sous-ensemble fini de label. On ne spécifie pas l'ensemble L mais on assume que les labels que l'on va utiliser seront disponibles. Cependant, on spécifie un sous-ensemble de $\mathbf{b}$ comme un **type de base d'arité** ce qui nous permet d'introduire la définition suivante :

Définition 1. *Un type de signature est un type de signature **d'arité** si et seulement si tous les types de base qu'il contient sont des types de base d'arité.*

Les ensembles finis peuvent être représentés par le type de signature $\sum_{l \in L} \alpha_l$ et les ensembles infinis sont représentés par le type de base correspondant.

Exemple 4. On peut représenter les booléens par $\sum_{bool \in \{true, false\}} 1$ et un sous-ensemble d'entier par $\sum_{n \in \{1, \ldots, m\}} 1$.

On définit les symboles de fonctions typés par : $\mathbf{f} : \alpha \to \beta$. Ils représentent les fonctions prédéfinies, telles que $+ : \mathbf{nat} \times \mathbf{nat} \to \mathbf{nat}$ ou encore $<: \mathbf{nat} \times \mathbf{nat} \to \mathbf{bool}$.

Pour finir, on définit les symboles d'opérations typés par : $\mathbf{op} : \alpha \to \beta$, tels que chaque β est un type de signature d'arité et tels que $\mathbf{op}$ a une théorie d'effet τ. Les opérations deviennent la source des effets et la théorie d'effet indique leurs propriétés.
On interprète $\mathbf{op} : \alpha \to \beta$ de la façon suivante : $\mathbf{op}$ accepte un paramètre α et après avoir exécuté l'effet approprié, son type de retour β détermine la continuation.

Maintenant que tout est définis, on peut créer une signature. Le choix de la signature va dépendre de ce que l'on veut représenter.

Exemple 5. On présente quelques exemple pris de [3].

> **Exceptions** : On prend l'unique symbole d'opération **raise** : **exc** $\to$ **0** paramétré sur l'ensemble **exc** afin de lever une exception. Le symbole de l'opération est nul (**0**) car il n'y a pas de continuation après une levée d'exception. Il faut donc gérer les exceptions.
>
> **State** : On prend le type de base d'arité **nat** (qui représente l'ensemble des naturelles), le type de base d'arité **loc** (un ensemble fini d'emplacement) et les symboles d'opérations **get** : **loc** $\to$ **nat** et **set** : **loc** $\times$ **nat** $\to$ **1**. On part du principe que l'on stocke des naturelles dans des emplacements mémoire, **get** permet de récupérer le naturelle pour un emplacement donné tandis que **set** remplace l'élément à l'emplacement donné en paramètre par le second élément donné en paramètre et ne retourne rien.

2.2 Types

Le langage que l'on définit suit l'approche *call-by-push-value* de Levy [4]. On a donc une séparation stricte entre les valeurs et les calculs. Ces types sont donnés par :

$$A, B := \mathbf{b} \mid \mathbf{1} \mid A \times B \mid \sum_{l \in L} A_l \mid U\underline{C}$$
$$\underline{C} := FA \mid \Pi_{l \in L}\ \underline{C}_l \mid A \to \underline{C}$$

Les types de valeur étendent les types de signature en ajoutant le constructeur de type $U_{\_}$. Le type $U\underline{C}$ représente les types de calcul $\underline{C}$ qui ont été bloqué dans une valeur. Le calcul pourra être forcé plus tard.

FA représente les calculs qui retourne une valeur de type A. La fonction $A \to \underline{C}$ représente les calculs de type $\underline{C}$ nécessitant un paramètre de type A. Pour finir, le type de calcul produit $\Pi_{l \in L}\underline{C}_l$ représente un ensemble fini indexé de calcul produit de type $\underline{C}_l$, pour $l \in L$. Ces tuples ne sont pas évalués de façon séquentiel comme dans la stratégie d'évaluation *call-by-value*. À la place, un composant d'un tuple est évalué seulement une fois qu'il a été sélectionné via une projection.

2.3 Termes

On a défini les signatures et les types, il nous manque les termes. On va les définir en trois parties : les termes **valeur** V, W, les termes **calcul** M, N et les termes **gestionnaire** H. Ils sont définit par :

$$\begin{aligned}
V, W &:= x \mid f(V) \mid \langle\rangle \mid \langle V, W\rangle \mid l(V) \mid \mathbf{thunk}\ M \\
M, N &:= \mathbf{match}\ V\ \mathbf{with}\ \langle x, y\rangle \mapsto M \mid \mathbf{match}\ V\ \mathbf{with}\ \{l(x_l) \mapsto M_l\}_{l \in L} \mid \mathbf{force}\ V \mid \\
&\quad\ \ \mathbf{return}\ V \mid M\ \mathbf{to}\ x : A.N \mid \langle M_l\rangle_{l\ \in L} \mid \mathbf{prj}_l\ M \mid \lambda x : A.M \mid M\ V \mid k(V) \mid \\
&\quad\ \ \mathbf{op}_V(x : \beta.M) \mid M\ \mathbf{handled\ with}\ H\ \mathbf{to}\ x : A.N \\
H &:= \{\mathbf{op}_{x:\alpha}(k : \beta \to \underline{C}) \mapsto M_{\mathbf{op}}\}_{\mathbf{op}:\alpha\to\beta}
\end{aligned}$$

avec x, y appartenant à l'ensemble des variables de valeur et k appartenant à l'ensemble des variables de continuation. $\{...\}_{l \in L}$ représente l'ensemble des calculs, un pour chaque $l \in L$. Même principe pour $\{...\}_{\mathbf{op}:\alpha\to\beta}$.

La syntaxe respecte celle présenté avec l'approche *call-by-push-value* excepté pour la partie gestionnaire et la dernière ligne des termes **calcul**.

On a le terme d'une *application d'opération*

$$\mathbf{op}_V(x : \beta.M)$$

Elle fonctionne de la manière suivante : l'opération **op** paramétré par V est effectuée, ensuite on affecte le résultat à x et enfin on évalue la continuation M.

Exemple 6. Le calcul suivant incrémente le nombre n, le stocke dans l'emplacement l et retourne l'ancienne valeur.

$$\mathbf{get}_l(n : \mathbf{nat}.\mathbf{set}_{\langle l, n+1\rangle}(x : \mathbf{1}.\mathbf{return\ n}))$$

Ensuite, le terme du gestionnaire

$$\{\mathbf{op}_{x:\alpha}(k:\beta \to \underline{C}) \mapsto M_{\mathbf{op}}\}_{\mathbf{op}:\alpha\to\beta}$$

est définit par un ensemble fini de définitions d'opérations qui gère les effets

$$\mathbf{op}_{x:\alpha}(k:\beta \to \underline{C}) \mapsto M_{\mathbf{op}}$$

une pour chaque **op**. Le terme gestionnaire $M_{\mathbf{op}}$ est dépendant de la variable x et de la continuation k. À noter que k sera toujours utiliser de la manière suivante : $k(V)$ ou V est la valeur appliqué à la continuation.

Exemple 7. Bien que l'on ne peut pas modifier une valeur en lecture-seule (c'est un cas spécifique ou on ne peut pas utiliser l'opération **set**), il est tout de même possible d'évaluer un calcul avec une valeur temporairement modifiée. Pour faire cela, on crée un gestionnaire *état-temporaire* qui dépend d'une variable n (la dite variable temporaire) :

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

Finalement, on a le terme de calcul *gestionnaire*

$$M \textbf{ handled with } H \textbf{ to } x : A.N$$

Il évalue M, gérant tous les calculs d'opérations grâce à H. Ensuite il associe le résultat à x et enfin évalue N. C'est un point clé pour la compréhension de l'article, on va donc détailler.

On suppose que M veut effectuer une opération $\mathbf{op}_V(y.M')$ correspondant au terme de gérant $\mathbf{op}_z(k) \mapsto M_{\mathbf{op}} \in H$. On substitue l'évaluation de **op** par $M_{\mathbf{op}}$ en associant z avec V et en remplaçant la continuation $k(W)$ pour chaque occurrence de celle-ci dans $M_{\mathbf{op}}$ avec

$$M'[W/y] \textbf{ handled with } H \textbf{ to } x : A.N$$

La continuation k reçoit la valeur W déterminé par $M_{\mathbf{op}}$. Elle est ensuite géré de la même façon que M, ce qui n'est pas le cas de $M_{\mathbf{op}}$. En effet $M_{\mathbf{op}}$ n'est pas géré par H, donc ses opérations échappent à H mais peuvent cependant être gérées par un autre gestionnaire imbriqué.

Exemple 8. On considère le gestionnaire d'état temporaire $H_{temporary}$ donné dans Exemple 7. On prend la calcul suivant :

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

Dans ce calcul on va associer n avec la valeur 20. Le premier **get** va être géré par $H_{temporary}$ et va donc associer x avec la valeur de n donc 20. Le second **get** appartient à la continuation et est donc géré aussi par $H_{temporary}$, i.e, il va associer y avec 20 (par la même logique que le premier **get**). On retourne $x + y$ donc $20 + 20$ soit 40 et on associe ce résultat à z. On retournera au final 42.

2.4 Jugement de types

Tous les jugements de types sont fait dans le contexte de valeurs

$$\Gamma = x_1 : A_1, ..., x_m : A_m$$

avec les variables de valeurs x_i associées à un type de valeurs A_i et dans le contexte de continuations

$$K = k_1 : \alpha_1 \to \underline{C}_1, ..., k_n : \alpha_n \to \underline{C}_n$$

avec les variables k_j associées à un type de continuation $\alpha_j \to \underline{C}_j$.

Les valeurs sont typés par $\Gamma \mid K \vdash V : A$, ensuite les calculs sont typés par $\Gamma \mid K \vdash M : \underline{C}$ et enfin les gestionnaires sont typés par $\Gamma \mid K \vdash H : \underline{C}$ **handler**.

Les valeurs sont typés en suivant les règles ci-dessous :

$$\Gamma \mid K \vdash x : A \ (x : A \in \Gamma) \qquad \frac{\Gamma \mid K \vdash V : \alpha}{\Gamma \mid K \vdash f(V) : \beta}\ (\mathbf{f} : \alpha \to \beta) \qquad \frac{\Gamma \mid K \vdash M : \underline{C}}{\Gamma \mid K \vdash \mathbf{thunk}\ M : U\underline{C}}$$

$$\frac{\Gamma \mid K \vdash V : A \qquad \Gamma \mid K \vdash W : B}{\Gamma \mid K \vdash \langle V, W \rangle : A \times B} \qquad \frac{\Gamma \mid K \vdash V : A_l}{\Gamma \mid K \vdash l(V) : \sum_{l \in L} A_l}\ (l \in L) \qquad \Gamma \mid K \vdash \langle\rangle : \mathbf{1}$$

Les calculs sont typés en suivant les règles ci-dessous :

$$\frac{\Gamma \mid K \vdash V : A \times B \qquad \Gamma\ x : A, y : B \mid K \vdash M : \underline{C}}{\Gamma \mid K \vdash \mathbf{match}\ V\ \mathbf{with}\ \langle x, y \rangle \mapsto M : \underline{C}}$$

$$\frac{\Gamma \mid K \vdash V : \sum_{l \in L} A_l \qquad \Gamma\ , x_l : A_l \mid K \vdash M_l : \underline{C}\ \ (l \in L)}{\Gamma \mid K \vdash \mathbf{match}\ V\ \mathbf{with}\ \{l(x_l) \mapsto M_l\}_{l \in L} : \underline{C}} \qquad \frac{\Gamma \mid K \vdash V : U\underline{C}}{\Gamma \mid K \vdash \mathbf{force}\ V : \underline{C}}$$

$$\frac{\Gamma \mid K \vdash M : FA \qquad \Gamma, x : A \mid K \vdash N : \underline{C}}{\Gamma \mid K \vdash M\ \mathbf{to}\ x : A.N : \underline{C}} \qquad \frac{\Gamma \mid K \vdash V : A}{\Gamma \mid K \vdash \mathbf{return}\ V : FA}$$

$$\frac{\Gamma \mid K \vdash V : \alpha \qquad \Gamma, x : \beta \mid K \vdash M : \underline{C}}{\Gamma \mid K \vdash \mathbf{op}_V(x : \beta.M) : \underline{C}}\ (\mathbf{op} : \alpha \to \beta) \qquad \frac{\Gamma \mid K \vdash M : \Pi_{l \in L}\underline{C}_l}{\Gamma \mid K \vdash prj\ M : \underline{C}_l}\ (l \in L)$$

$$\frac{\Gamma \mid K \vdash M : A \to \underline{C} \qquad \Gamma \mid K \vdash V : A}{\Gamma \mid K \vdash M\ V : \underline{C}} \qquad \frac{\Gamma, x : A \mid K \vdash M : \underline{C}}{\Gamma \mid K \vdash \lambda x : A.M : A \to \underline{C}}$$

$$\frac{\Gamma \mid K \vdash V : \alpha}{\Gamma \mid K \vdash k(V) : \underline{C}}\ (k : \alpha \to \underline{C} \in K) \qquad \frac{\Gamma \mid K \vdash M_l : \underline{C}_l\ \ (l \in L)}{\Gamma \mid K \vdash \langle M_l \rangle_{l\ \in L} : \Pi_{l \in L}\underline{C}_l}$$

$$\frac{\Gamma \mid K \vdash M : FA \qquad \Gamma \mid K \vdash H : \underline{C}\ \mathbf{handler} \qquad \Gamma, x : A \mid K \vdash N : \underline{C}}{\Gamma \mid K \vdash M\ \mathbf{handled\ with}\ H\ \mathbf{to}\ x : A.N : \underline{C}}$$

Les gestionnaires sont typés en suivant la règle ci-dessous :

$$\frac{\Gamma, x : \alpha \mid K, k : \beta \to \underline{C} \vdash M_{\mathbf{op}} : \underline{C}\ \ (\mathbf{op} : \alpha \to \beta)}{\Gamma \mid K \vdash \{\mathbf{op}_{x:\alpha}(k : \beta \to \underline{C}) \mapsto M_{\mathbf{op}}\}_{\mathbf{op}:\alpha\to\beta} : \underline{C}\ \mathbf{handler}}$$

À partir de maintenant, on utilisera une succession d'abréviations décrite en Annexe.

Définition 2. *Quand un gestionnaire contient des termes gérant uniquement un sous-ensemble Θ de symbole d'opérations, on assume que les symboles d'opérations restants sont gérés par "eux-même". De façon plus formel, on va définir le gestionnaire tel que :*

$$\{\boldsymbol{op}_x(k) \mapsto M_{\boldsymbol{op}}\}_{\boldsymbol{op} \in \Theta} \overset{def}{=} \left\{ \boldsymbol{op}_x(k) \mapsto \left\{ \begin{array}{ll} M_{\boldsymbol{op}} & (\boldsymbol{op} \in \Theta) \\ \boldsymbol{op}_x(y : \beta.k(y)) & (\boldsymbol{op} \notin \Theta) \end{array} \right. \right\}_{\boldsymbol{op}}$$

3 Exemples

On va montrer à travers une succession d'exemples la portée des gestionnaires d'effets algébriques. La notion de validité des exemples est traitée plus loin dans le résumé.

3.1 Non-déterminisme explicite

L'évaluation d'un calcul non-déterminisme prend habituellement un seul chemin possible. Une alternative est de prendre tous les chemins possibles et d'autoriser qu'un chemin échoue. On utilisera cette alternative comme en Haskell. Pour cela, on ajoute un symbole d'opération **fail** (qui ne prend aucun paramètre) afin de représenter les chemins qui échouent.

On considère un gestionnaire qui introduit le résultat d'un calcul dans une liste. On définit des listes de types basiques **b** par le symbole $\mathbf{list_b}$. Ainsi que les symboles de fonctions: $\mathbf{nil} : \mathbf{1} \to \mathbf{list_b}$, $\mathbf{cons} : \mathbf{b} \times \mathbf{list_b} \to \mathbf{list_b}$ et $\mathbf{append} : \mathbf{list_b} \times \mathbf{list_b} \to \mathbf{list_b}$.

On définit un calcul qui pour tout élément de type **b** retourne un élément de type $\mathbf{list_b}$ tel que :

$$\Gamma \mid K \ \vdash \ M \ \mathbf{handled\ with}\ H_{list}\ \mathbf{to}\ x : \mathbf{b}.\mathbf{return\ cons}(x, \mathbf{nil}) : F\mathbf{list_b}$$

avec H_{list} définit par :

$$\begin{aligned}
\Gamma \mid K \ \vdash \ \{ & \\
& \mathbf{fail}() \mapsto \mathbf{return\ nil}, \\
& \mathbf{choose}(k_1, k_2) \mapsto k_1\ \mathbf{to}\ l_1 : \mathbf{list_b}.k_2\ \mathbf{to}\ l_2 : \mathbf{list_b}.\mathbf{return\ append}(l_1, l_2) \\
& \} : F\mathbf{list_b}\ \mathbf{handler}
\end{aligned}$$

Voyons comment le gestionnaire fonctionne.

(1) On considère que M contient un symbole d'opération **choose**. Lorsque l'on va évaluer l'opération $\mathbf{choose}(x, y)$, on va regarder si M est géré. Ici, on a H_{list}. Ensuite on va voir dans H_{list} si **choose** est géré. C'est le cas, on va donc calculer

$$M_{\mathbf{choose}}[x/k1, y/k2] = (k_1\ \mathbf{to}\ l_1 : \mathbf{list_b}.k_2\ \mathbf{to}\ l_2 : \mathbf{list_b}.\mathbf{return\ append}(l_1, l_2))[x/k1, y/k2]$$

Deux choses à noter : la continuation n'est pas utilisé dans $M_{\mathbf{choose}}$; on part du principe que les continuations k_1 et k_2 retourne un élément de type $\mathbf{list_b}$.

(2) On considère que M contient un symbole d'opération **fail**. En suivant la même logique que ci-dessus, On va calculer

$$M_{\mathbf{fail}} = \mathbf{return\ nil}$$

Une fois encore la continuation n'est pas gardé. Il est important de noter qu'un chemin invalide retourne aussi un élément de type $\mathbf{list_b}$.

3.2 Timeout

Le gestionnaire du *timeout* donne un exemple de paramètre passé dans la continuation. On souhaite effectuer un calcul et attendre pour un certains temps donné. Si le calcul n'est pas compléter avant le temps donné alors on retourne une valeur par défaut. On représente le temps via une seule opération $\mathbf{delay} : \mathbf{nat} \to \mathbf{1}$, telle que $\mathbf{delay}_t(M)$ est le calcul qui attend un temps t et ensuite effectue le calcul M. Pour tout type A, on a le gestionnaire $H_{timeout}$ suivant:

$$\begin{aligned}
\Gamma, x_0 : A, t_{wait} : \mathbf{nat} \mid K \ \vdash \ \{ & \\
& \mathbf{delay}_t(k : \mathbf{nat} \to FA) \mapsto \lambda t_{spent} : \mathbf{nat}. \\
& \mathbf{if}\ t + t_{spent} \leq t_{wait} \\
& \quad \mathbf{then\ delay}_t(k(t + t_{spent})\ t_{spent}) \\
& \quad \mathbf{else\ delay}_{t_{wait} - t_{spent}}(\mathbf{return}\ x_0) \\
& \} : \mathbf{nat} \to FA\ \mathbf{handler}
\end{aligned}$$

Le terme qui gère **delay** (que l'on va nommé $M_{\mathbf{delay}}$) dépend de la variable t_{spent}, qui représente le temps que l'on a déjà attendu. Si l'on souhaite utiliser ce gestionnaire, on va devoir lui spécifier un paramètre initial. On veut gérer $M : FA$ via $H_{timeout}$ or on a le calcul suivant :

$$\Gamma, x_0 : A, t_{wait} : \mathbf{nat} \mid K \vdash M \textbf{ handled with } H_{timeout} \textbf{ to } x : A.\lambda t : \mathbf{nat}.\mathbf{return}\ x : \mathbf{nat} \to FA$$

Le calcul géré est de type $\mathbf{nat} \to FA$, on voit la nécessité du paramètre initial pour passer à un calcul géré de type FA.

$$\Gamma, x_0 : A, t_{wait} : \mathbf{nat} \mid K \vdash (M \textbf{ handled with } H_{timeout} \textbf{ to } x : A.\lambda t : \mathbf{nat}.\mathbf{return}\ x)\ 0 : FA$$

il est nécessaire d'ajouter une abstraction autour de **return** x car si on suppose que M ne contient pas de symbole d'opération **delay** alors on aura le calcul suivant : $(\lambda t : \mathbf{nat}.\mathbf{return}\ x)\ 0 : FA$.

Exemple 9. On effectue le calcul ci-dessous avec $M = \mathbf{delay}_5(t_2.(\mathbf{return}\ t_2 + 45))$ (on symbolisera le passage d'une étape de calcul via le symbole $\Rightarrow$ et les substitutions par $[x/y]$ où x remplace y):

$$\begin{aligned}
&\lambda t_{wait}.(\lambda x_0.((M \textbf{ handled with } H_{timeout} \textbf{ to } x.\lambda t.\mathbf{return}\ x)\ 0)\ 4)\ 20 \\
\Rightarrow\ &\lambda x_0.((M \textbf{ handled with } H_{timeout} \textbf{ to } x.\lambda t.\mathbf{return}\ x)\ 0)\ 4\ [20/t_{wait}] \\
\Rightarrow\ &(M \textbf{ handled with } H_{timeout} \textbf{ to } x.\lambda t.\mathbf{return}\ x)\ 0\ [20/t_{wait}, 4/x_0]
\end{aligned}$$

Rappel : Pour évaluer un calcul géré, on évalue en premier M; ensuite on associe la valeur de retour de M à x et enfin on évalue $\lambda t.\mathbf{return}\ x$.

Pour continuer on substitue M.

$$\Rightarrow (\mathbf{delay}_5(t_2.(\mathbf{return}\ t_2 + 45)) \textbf{ handled with } H_{timeout} \textbf{ to } x.\lambda t.\mathbf{return}\ x)\ 0\ [20/t_{wait}, 4/x_0]$$

delay est un symbole d'opération géré par $H_{timeout}$. On va donc prendre le terme gérant $M_{\mathbf{delay}}$. Pour alléger l'écriture des substitutions ci-après on définit *conti* telle que

$$\mathit{conti} = (\mathbf{return}\ t_2 + 45)[t + t_{spent}/t2] \textbf{ handled with } H_{timeout} \textbf{ to } x.\lambda t.\mathbf{return}\ x\ [20/t_{wait}, 4/x_0]$$

On se retrouve avec le calcul suivant :

$$\Rightarrow (\lambda t_{spent}.\mathbf{if}\ t + t_{spent} \le t_{wait} \mathbf{\ then\ delay}_t(k(t + t_{spent})\ t_{spent}) \mathbf{\ else\ delay}_{t_{wait} - t_{spent}}(\mathbf{return}\ x_0)$$
$$[\mathit{conti}/k(t + t_{spent}), 5/t])\ 0\ [20/t_{wait}, 4/x_0]$$

$$\Rightarrow (\lambda t_{spent}.\mathbf{if}\ 5 + t_{spent} \le 20 \mathbf{\ then\ delay}_5(k(5 + t_{spent})\ t_{spent}) \mathbf{\ else\ delay}_{20 - t_{spent}}(\mathbf{return}\ 4)$$
$$[\mathit{conti}/k(5 + t_{spent}), 5/t])\ 0\ [20/t_{wait}, 4/x_0]$$

$$\Rightarrow \mathbf{if}\ 5 \le 20 \mathbf{\ then\ delay}_5(k(5)\ 0) \mathbf{\ else\ delay}_{20}(\mathbf{return}\ 4)\ [0/t_{spent}, \mathit{conti}/k(5), 5/t, 20/t_{wait}, 4/x_0]$$
$$\Rightarrow \mathbf{delay}_5(k(5)\ 0)[\mathit{conti}/k(5), 20/t_{wait}, 4/x_0]$$

À ce niveau là, **delay** n'est pas géré donc on effectue juste l'opération.

$$\begin{aligned}
\Rightarrow\ &k(5)\ 0\ [\mathit{conti}/k(5), 20/t_{wait}, 4/x_0] \\
\Rightarrow\ &(\mathbf{return}\ 5 + 45 \textbf{ handled with } H_{timeout} \textbf{ to } x.\lambda t.\mathbf{return}\ x)\ 0\ [20/t_{wait}, 4/x_0] \\
\Rightarrow\ &(\lambda t.\mathbf{return}\ x)\ 0\ [50/x, 20/t_{wait}, 4/x_0] \\
\Rightarrow\ &\mathbf{return}\ 50\ [0/t, 50/x, 20/t_{wait}, 4/x_0]
\end{aligned}$$

3.3 Rollback

Quand un calcul provoque une levée d'exception, il est primordial d'annuler toute modification faite dans la mémoire. Ce comportement est nommé *rollback*. On va supposer que l'on a qu'un seul emplacement (pour plus de simplicité) l_0. Un gestionnaire approprié pour le *rollback*, $H_{rollback}$, est définit par :

$$\Gamma, n_{init} : \mathbf{nat} \mid K \ \vdash \ \{\mathbf{raise}_e(k) \mapsto \mathbf{set}_{l_0,n_{init}}(M' \ e)\} : \underline{C} \ \mathbf{handler}$$

où $M' : \mathbf{exc} \to \underline{C}$. On doit donner la valeur initial de l'emplacement l_0 pour le retour en arrière. Pour cela, on définit un calcul géré par $H_{rollback}$ par :

$$\Gamma \mid K \ \vdash \ \mathbf{get}_{l_0}(n_{init} : \mathbf{nat}.M \ \mathbf{handled\ with} \ H_{rollback}) : \underline{C}$$

Une alternative est un gestionnaire avec passage de paramètre, qui ne modifie pas la mémoire, mais garde la trace de tous les changements effectués dans M. Si aucune exception est levée alors on met à jour l'emplacement. Ce gestionnaire $H_{param-rollback}$ est défini par :

$$\begin{array}{l} \Gamma \mid K \ \vdash \ \{ \\ \qquad \mathbf{get}_{l_0}(k : \mathbf{nat} \to (\mathbf{nat} \to \underline{C})) \mapsto \lambda n : \mathbf{nat}.k(n) \ n \\ \qquad \mathbf{set}_{l_0,n'}(k : \mathbf{1} \to (\mathbf{nat} \to \underline{C})) \mapsto \lambda n : \mathbf{nat}.k() \ n' \\ \qquad \mathbf{raise}_e() \mapsto \lambda n : \mathbf{nat}.M' \ e \\ \qquad \} : \mathbf{nat} \to \underline{C} \ \mathbf{handler} \end{array}$$

Il est utilisable sur le calcul M suivant :

$$\Gamma \mid K \ \vdash \ \mathbf{get}_{l_0}(n_{init} : \mathbf{nat}.(M \ \mathbf{handled\ with} \ H_{rollback} \ \mathbf{to} \ x : A.\lambda n : \mathbf{nat}.\mathbf{set}_{l_0,n}(\mathbf{return} \ x)) \ n_{init})$$

On récupère la valeur initial de l'emplacement l_0 que l'on associe à la variable n_{init}. Si aucune exception n'est levée alors on met à jour l'emplacement l_0.

4 Sémantique

On va introduire la théorie des effets. La partie interprétation sera omit dans ce résumé car il fait appel à des notions de catégorie et des notions poussées sur les modèles. Je n'ai pas la présomption d'être en capacité de le comprendre entièrement et de l'expliquer.

4.1 La théorie des effets

On va décrire les propriétés des effets avec des équation entre patrons T. Ils décrivent la forme général de tous les calculs sans présumer des types. On définit T par :

$$T := z(V) \mid \mathbf{match} \ V \ \mathbf{with} \ \langle x, y \rangle \mapsto T \mid \mathbf{match} \ V \ \mathbf{with} \ \{l(x_l) \mapsto T_l\}_{l \in L} \mid \mathbf{op}_V(x : \beta.T)$$

avec z appartenant à un ensemble de variable de patron. Dans les patrons, on se limite à la signature des valeurs. Ceux sont des valeurs qui peuvent être typées par $\Gamma \vdash V : \alpha$, avec Γ le contexte des variables de valeurs liées à un type de signature. Les règles de typage sont, en omettant le contexte de continuation et les règles pour les types bloqués, les même que ceux pour leurs valeurs.

On construit des patrons dans le contexte de variables de valeur, liée à un type de signature, et un contexte de patron

$$Z = z_1 : \alpha_1, ..., z_n : \alpha_n$$

avec z_j une variable de patron liée à un type de signature d'arité α_j. $z_j : \alpha_j$ ne représente pas une valeur de type α_j mais un calcul qui dépend d'une telle valeur. Le jugement de typage pour un patron bien-formé est $\Gamma \mid Z \vdash T$ et est donné par les règles suivantes :

$$\frac{\Gamma \vdash V : \alpha}{\Gamma \mid Z \vdash z(V)} \ (z : \alpha \in Z) \qquad \frac{\Gamma \vdash V : A \times B \qquad \Gamma\ x : \alpha, y : \beta \mid Z \vdash T}{\Gamma \mid Z \vdash \textbf{match}\ V\ \textbf{with}\ \langle x, y\rangle \mapsto T}$$

$$\frac{\Gamma \vdash V : \sum_{l \in L} \alpha_l \qquad \Gamma\ , x_l : \alpha_l \mid Z \vdash T_l \ \ (l \in L)}{\Gamma \mid Z \vdash \textbf{match}\ V\ \textbf{with}\ \{l(x_l) \mapsto T_l\}_{l \in L}} \qquad \frac{\Gamma \vdash V : \alpha \qquad \Gamma, x : \beta \mid Z \vdash T}{\Gamma \mid Z \vdash \mathbf{op}_V(x : \beta.T)} \ (\mathbf{op} : \alpha \to \beta)$$

Une théorie d'effet τ est un ensemble fini d'équation $\Gamma \mid Z \vdash T_1 = T_2$, avec T_1 et T_2 bien formé par rapport à Γ et Z.

Exemple 10. On reprend les symboles d'opérations de l'Exemple 5.

Exceptions: La théorie d'effet est un ensemble vide, car les exceptions ne satisfont aucune équations non-trivial.

États: La théorie d'effet est définie par les équations suivantes (on n'écrit pas les contextes pour une meilleur lecture) :

$$\begin{aligned}
\mathbf{get}_l(x.z) &= z \\
\mathbf{get}_l(x.\mathbf{set}_{l,x}(z)) &= z \\
\mathbf{set}_{l,x}(\mathbf{set}_{l,x'}(z)) &= \mathbf{set}_{l,x'}(z) \\
\mathbf{set}_{l,x}(\mathbf{get}_l(x'.z(x'))) &= \mathbf{set}_{l,x}(z(x)) \\
\mathbf{get}_x(x.\mathbf{get}_l(x'.z(x,x'))) &= \mathbf{get}_l(x.z(x,x)) \\
\mathbf{set}_{l,x}(\mathbf{set}_{l',x'}(z)) &= \mathbf{set}_{l',x'}(\mathbf{set}_{l,x}(z)) && (l \neq l') \\
\mathbf{set}_{l,x}(\mathbf{get}_{l'}(x'.z(x'))) &= \mathbf{get}_{l'}(x'.\mathbf{set}_{l,x}(z(x'))) && (l \neq l') \\
\mathbf{get}_l(x.\mathbf{get}_{l'}(x'.z(x,x'))) &= \mathbf{get}_{l'}(x'.\mathbf{get}_l(x.z(x,x'))) && (l \neq l')
\end{aligned}$$

On a simplifié l'écriture des trois dernières équations. En effet on a ajouté sur le côté la condition $(l \neq l')$. Cela reste dans le cadre de la définition de la théorie d'effet, $T_1 = T_2$ $(l \neq l')$ abrège l'équation suivante :

$$T_1 = (\textbf{if}\ l \neq_{\mathbf{loc}} l'\ \textbf{then}\ T_2\ \textbf{else}\ T_1)$$

Non-déterminisme: La théorie d'effet est définie par les équations suivants :

$$\begin{aligned}
\mathbf{choose}(z, z) &= z \\
\mathbf{choose}(z_1, z_2) &= \mathbf{choose}(z_2, z_1) \\
\mathbf{choose}(z_1, \mathbf{choose}(z_2, z_3)) &= \mathbf{choose}(\mathbf{choose}(z_1, z_2), z_3)
\end{aligned}$$

Temps: La théorie d'effet est définie par les équations suivants :

$$\begin{aligned}
\mathbf{delay}_0(z) &= z \\
\mathbf{delay}_{t_1}(\mathbf{delay}_{t_2}(z)) &= \mathbf{delay}_{t_1+t_2}(z)
\end{aligned}$$

Exception destructrice: La signature des exceptions destructives est l'union de la signature de l'état et de l'exception. La théorie d'effet comprend toute les équations de l'état plus les équations suivantes :

$$\begin{aligned}
\mathbf{get}_l(x.\mathbf{raise}_e()) &= \mathbf{raise}_e() \\
\mathbf{set}_{l,x}(\mathbf{raise}_e()) &= \mathbf{raise}_e()
\end{aligned}$$

Comme **raise** n'évalue jamais la continuation, on écrit $\mathbf{raise}_e()$ à la place de $\mathbf{raise}_e(x : \mathbf{0}.T)$.

Les équations ci-dessus implique que les opérations sur la mémoire suivit par une levée d'exception est la même chose que juste lever une exception. Cela implique que toutes les opérations sur la mémoire sont discutables si une exception apparaît. La théorie des exceptions destructives est discutée comme celle du "rollback" dans [3].

5 Raisonnement à propos des gestionnaires

On s'intéresse à deux questions en particulières : Quels calculs sont égaux et quels gestionnaires sont corrects ? On va se contenter dans cette section des observations initiales sans donner une logique complète.

On fixe une signature et une théorie d'effet τ avec ses interprétations. On considère une formule bien formée $\Gamma \mid K \vdash \varphi$. Elle est créée à partir de formules atomiques grâce aux connecteurs booléens habituelles ainsi que les quantificateurs existentielle et universelle appliqués sur les types de valeurs (ex : $\forall x : A.\varphi$) et sur les continuations (ex : $\forall k : \alpha \to \underline{C}.\varphi$).

Remarque. Dans la suite on utilisera l'égalité, l'inégalité de Kleene ainsi que les assertions d'existence. On introduit donc les notations ainsi que leurs signification dans cette remarque.

assertion d'existence: on note $e\downarrow$ l'assertion et elle est vraie si et seulement si l'expression e est définie.

égalité de kleene: on la note $e \simeq e'$ et elle vrai si les expressions sont définies et équivalente ou les expressions sont indéfinies.

inégalité de kleene: on la note $e \lesssim e'$ et elle abrège la notation $e\downarrow \;\Rightarrow\; e \simeq e'$

On va appliquer ses assertions sur les termes de valeurs et de calculs. On pourra interprété $H\downarrow$ comme la validité du gestionnaire.

Exemple 11. On considère un ensemble d'équations qui garde la syntaxe et la sémantique évoqué jusqu'alors. On peut définir une β-inéquation de Kleene avec la construction suivante :

$$\textbf{return } x \textbf{ to } x : A.M \lesssim M$$

et l'η-équation pour la fonction suivante : $\lambda x : A.M\ x \simeq M$

On définit des équations qui décrivent les comportements des opérations sur les types de produits et de fonctions :

$$\mathbf{op}_V(x : \alpha.\langle M_l\rangle_{l\ \in L}) \simeq \langle \mathbf{op}_V(x : \alpha.M_l)\rangle_{l\ \in L} : \Pi_{l\in L}\underline{C}_l$$
$$\mathbf{op}_V(x : \alpha.\lambda y : A.M) \simeq \lambda y : A.\mathbf{op}_V(x : \alpha.M_l) : A \to \underline{C}$$

On a aussi une équation qui définit la commutation entre les opérations et le séquençage:

$$\mathbf{op}_V(x : \alpha.M) \textbf{ to } y : A.N \simeq \mathbf{op}_V(x : \alpha.M \textbf{ to } y : A.N) : \underline{C}$$

D'autres équations sont héritées directement de la théorie d'effet τ. On prend le patron T tel que $x_1 : \alpha_1, ..., x_m : \alpha_m | z_1 : \beta_1, ..., z_n : \beta_n \vdash T$ et les variables de continuations distinctes $k_1, ..., k_n$. On réutilise la notation $T[k_1/z_1, ..., k_n/z_n]$ qui représente le terme de calcul obtenu après avoir substitué toutes les variables z_i par leurs continuations associées k_i.

Si on prend *Gamma* le contexte de valeurs contenant $x_1, \alpha_1, ..., x_m : \alpha_m$ et K le contexte des continuations contenant $k_1 : \beta_1 \to \underline{C}, ..., k_n : \beta_n \to \underline{C}$ **alors** on peut typer les patrons par $\Gamma \mid K \vdash T[k_1/z_1, ..., k_n/v_n] : \underline{C}$. On peut donc avoir :

$$T_1[k_1/z_1, ..., k_n/v_n] \simeq T_2[k_1/z_1, ..., k_n/v_n]$$

pour toutes équations

$$x_1 : \alpha_1, ..., x_m : \alpha_m | z_1 : \beta_1, ..., z_n : \beta_n \vdash T_1 = T_2$$

dans τ et pour tout $k_1 : \beta_1 \to \underline{C}, ..., k_n : \beta_n \to \underline{C}$.

On définit deux inéquations pour la construction des gestionnaires. Pour tout $H = \mathbf{op}_y(k) \mapsto M_{\mathbf{op}_{\mathbf{op}:\alpha\to\beta}}$, les inéquations sont :

$$\begin{aligned}\mathbf{return}\ x\ \mathbf{handled\ with}\ H\ \mathbf{to}\ x : A.N &\lesssim N\\ \mathbf{op}_y(x' : \beta.M)\ \mathbf{handled\ with}\ H\ \mathbf{to}\ x : A.N &\lesssim M_{\mathbf{op}}[x' : \beta.M\ \mathbf{handled\ with}\ H\ \mathbf{to}\ x : A.N/k]\end{aligned}$$

où la substitution de la forme $x' : \beta.M'/k$ remplace chaque occurrence du calcul $k(W)$ par $M'[V/x']$. On a le cas spécifique suivant

$$M\ \mathbf{to}\ x : A.N \simeq M \mathbf{handled\ with}\ \{\}\ \mathbf{to}\ x : A.N$$

qui montre que le séquençage est un cas spécifique du gestionnaire où toutes les opérations sont gérées par elles-même.

Il y a d'autres équations pour les gestionnaires d'exceptions proposé par Benton et Kennedy [2] et par Levy [5]. On souhaiterai généraliser ces équations à nos gestionnaires. Il se trouve que ces équations échouent pour les gestionnaires généraux, cependant elles sont correctes pour certaines classes de gestionnaires (voir [6]). Par la suite on ne prendra pas en compte ce problème.

On considère les assertions d'existences suivantes, elles définissent toutes les conditions pour une interprétation d'un terme d'exister.

$$\begin{aligned}\mathbf{return}\ V \downarrow &\Leftrightarrow\ V\downarrow\\ M\ \mathbf{to}\ x : A.N \downarrow &\Leftrightarrow M\downarrow\ \wedge\ \forall x : A.N\downarrow\\ \lambda x : A.M \downarrow &\Leftrightarrow\ \forall x : A.M\downarrow\\ \mathbf{op}_V(x : \alpha.M)\downarrow &\Leftrightarrow\ V\downarrow\ \wedge\ \forall x : \alpha.M\downarrow\end{aligned}$$

Afin de définir l'existence d'un gestionnaire, on doit d'abord être capable de remplacer les opérations dans des patrons par leurs définitions dans le gestionnaire.
On définit le gestionnaire $\Gamma \mid H \vdash: \underline{C}$ **handler**, avec $H = \{\mathbf{op}_{x:\alpha}(k : \beta \to \underline{C}) \mapsto M_{\mathbf{op}}\}_{\mathbf{op}:\alpha\to\beta}$ et un contexte de variable de patron Z. Ensuite on prend K' le contexte de continuation tel que $\forall z : \alpha \in Z\ :\ k_z : \alpha \to \underline{C}$. Enfin, pour tout les termes de patrons $\Gamma' \mid Z \vdash T$ on définit récursivement les termes de calculs $\Gamma, \Gamma' \mid K, K' \vdash T^H : \underline{C}$ par :

$$\begin{aligned}z(V)^H &= k_z(V)\\ (\mathbf{match}\ V\ \mathbf{with}\ \langle x_1, x_2\rangle \mapsto T)^H &= \mathbf{match}\ V\ \mathbf{with}\ \langle x_1, x_2\rangle \mapsto T^H\\ (\mathbf{match}\ V\ \mathbf{with}\ \{l(x_l) \mapsto T_l\}_{l\in L})^H &= \mathbf{match}\ V\ \mathbf{with}\ \{l(x_l) \mapsto T_l^H\}_{l\in L}\\ \mathbf{op}_V(y : \beta.T)^H &= M_{\mathbf{op}}[V/x, y : \beta.T^H/k]\end{aligned}$$

On a l'équivalence suivante

$$H\downarrow\ \Leftrightarrow\ \bigwedge\{\forall x : \alpha.M_{\mathbf{op}}\downarrow\ \mid \mathbf{op} : \alpha \to \beta\}\ \wedge\ \bigwedge\{T_1^H \simeq T_2^H\ \mid\ \Gamma' \mid Z \vdash T_1 = T_2 \in \tau\}$$

qui affirme que le gestionnaire est correct si les opérations sont définit et respect les équations de la théorie des effets. On regarde maintenant la construction du gestionnaire en elle-même et on remarque l'équivalence suivante :

$$M\ \mathbf{handled\ with}\ H\ \mathbf{to}\ x : A.N\downarrow\ \Leftrightarrow\ M\downarrow\ \wedge\ H\downarrow\ \wedge\ \forall x : A.N\downarrow$$

6 Validité des gestionnaires

Décider de la validité d'un gestionnaire est une question intéressante qui peut être pertinente pour les développeur de compilateur. Dans cette section, on va montrer quelques résultat sans donner leurs preuves.

Définition 3. *Un gestionnaire* $\Gamma \mid K \vdash H : \underline{C}$ ***handler*** *est* ***simple*** *si*

- *en réorganisant,* Γ *a la forme*

$$x_1 : \alpha_1, ..., x_m : \alpha_m, f_1 : U(\beta_1 \to \underline{C}), ..., f_n : U(\beta_n \to \underline{C})$$

- *en réorganisant,* K *a la forme*

$$k'_1 : \beta'_1 \to \underline{C}, ..., k'_p : \beta'_p \to \underline{C}$$

- *et pour tout* $\boldsymbol{op} : \alpha \to \beta$*, il y a un patron*

$$x_1 : \alpha_1, ..., x_m : \alpha_m, x : \alpha \mid z_1 : \beta_1, ..., z_n : \beta_n, z'_1 : \beta'_1, ..., z'_p : \beta'_p, z : \beta \vdash T_{\boldsymbol{op}}$$

tel que le terme gérant

$$\Gamma, x : \alpha \mid K, k : \beta \to \underline{C} \vdash M_{\boldsymbol{op}} : \underline{C}$$

est obtenu par la substitution de $T_{\boldsymbol{op}}$ *qui remplace tous les* x_i *par eux-même,* x *par lui-même, tous les* z_j *par* $(y_j : \beta_j.(\textbf{\textit{force}}\ f_j)\ y_j)$*, tous les* z'_l *par* $(y'_l : \beta'_l.k'_l(y'_l))$ *et* z *par* $(y : \beta.k(y))$*.*

Le gestionnaire $H_{Temporary}$ (Exemple 7) est **simple**, aucun des gestionnaires avec des paramètres passés sont simple car ils contiennent tous une lambda abstraction dans leur terme gérant. Le gestionnaire d'exception

$$\{\mathbf{raise}_{y:exc}(k : \mathbf{0} \to \underline{C}) \mapsto \mathbf{match}\ y\ \mathbf{with}\ \{e(z) \mapsto N_e\}_{e \in \mathbf{exc}}\}$$

n'est pas simple. Toutefois, on peut utiliser le gestionnaire simple

$$f : U(\mathbf{exc} \to \underline{C}) \vdash \{\mathbf{raise}_{e:\mathbf{exc}}(k : \mathbf{0} \to \underline{C}) \mapsto (\mathbf{force}\ f)\ e\}$$

avec f la fonction bloquée tel que

$$f = \lambda y : \mathbf{exc}.\mathbf{match}\ y\ \mathbf{with}\ \{e(z) \mapsto N_e\}_{e \in \mathbf{exc}}$$

Ce gestionnaire a le même comportement que celui définit plus haut.

Remarque. Une signature est **simple** si elle n'a pas de types de base et pas de symboles de fonction. Les signatures données pour les exceptions est sont simple mais pas pour les états.

Théorème 1. *Le problème de décision, en prenant une signature simple, une théorie d'effet, un simple gestionnaire clos* $\vdash H : F\mathbf{0}$ ***handler****, sachant que le gestionnaire est correct, est* Π_2*-**complet**.*

La nature polymorphique des variables de patron signifie que les patrons peuvent définir une famille entière de gestionnaire, une pour chaque type de calcul. Cela nous amène à la définition d'une famille de gestionnaire uniformément simple.

Définition 4. *Une famille de gestionnaire* $\{\Gamma_{\underline{C}} \mid K_{\underline{C}} \vdash H_{\underline{C}} : \underline{C}\ \textbf{\textit{handler}}\}_{\underline{C}}$*, avec* $\underline{C}$ *l'ensemble des types de calcul, est* ***uniformément simple*** *si*

- *en réorganisant,* $\Gamma_{\underline{C}}$ *a la forme*

$$x_1 : \alpha_1, ..., x_m : \alpha_m, f_1 : U(\beta_1 \to \underline{C}), ..., f_n : U(\beta_n \to \underline{C})$$

- *en réorganisant,* $K_{\underline{C}}$ *a la forme*

$$k'_1 : \beta'_1 \to \underline{C}, ..., k'_p : \beta'_p \to \underline{C}$$

- *et pour tout* $\boldsymbol{op} : \alpha \to \beta$*, il y a un patron*

$$x_1 : \alpha_1, ..., x_m : \alpha_m, x : \alpha \mid z_1 : \beta_1, ..., z_n : \beta_n, z'_1 : \beta'_1, ..., z'_p : \beta'_p, z : \beta \vdash T_{\boldsymbol{op}}$$

tel que le terme gérant

$$\Gamma_{\underline{C}}, x : \alpha \mid K_{\underline{C}^{\prime}}, k : \beta \to \underline{C} \vdash M_{\boldsymbol{op}} : \underline{C}$$

est obtenu par la substitution de $T_{\boldsymbol{op}}$ *qui remplace tous les* x_i *par eux-même,* x *par lui-même, tous les* z_j *par* $(y_j : \beta_j.(\textbf{\textit{force}}\ f_j)\ y_j)$*, tous les* z'_l *par* $(y'_l : \beta'_l.k'_l(y'_l))$ *et* z *par* $(y : \beta.k(y))$*.*

Il est évident que tous les gestionnaires du famille uniformément simple sont simple. Corollairement, tous les gestionnaires simple, via l'aspect polymorphique des variables de patron, peuvent définir une famille uniformément simple. Dans ceux vu en amont, le gestionnaire $H_{Temporary}$ est uniformément simple. Tous ceux qui sont définit pour un type de calcul précis (ex **F0**) ne sont pas uniformément simple.

Définition 5. *Une famille de gestionnaires* $\{\Gamma_{\underline{C}} \mid K_{\underline{C}} \vdash H_{\underline{C}} : \underline{C}$ ***handler***$\}_{\underline{C}}$ *pour une signature donnée, avec* $\underline{C}$ *l'ensemble des calculs, est* ***correct*** *si chaque gestionnaire de la famille est correct.*

Sachant qu'une famille de gestionnaires uniformément simple ne peut utiliser les propriétés d'un type de calcul spécifique, cela ne peut pas être aussi artificiel qu'une famille de gestionnaire arbitraire. On peut cependant s'attendre à ce que la preuve de la validité soit plus simple. La validité peut devenir **semi-décidable**.

Théorème 2. *Le problème de décision, avec une signature donnée, une théorie d'effet et une famille de gestionnaires* $\{\vdash H_{\underline{C}} : \underline{C}$ ***handler***$\}_{\underline{C}}$ *close uniformément simple, sachant que la famille est correct, est* $\sum_1$-***complet****.*

Les théories d'effet, avec une signature simple, correspondent à des théories d'équations finies ordinaire. Ceci étant, on peut transférer la notion de décidabilité sur eux. Avec ça on peut dire :

Théorème 3. *Le problème de décision, avec une signature donnée, une théorie d'effet décidable et une famille de gestionnaires* $\{\vdash H_{\underline{C}} : \underline{C}$ ***handler***$\}_{\underline{C}}$ *close uniformément simple, sachant que la famille est correct, est* ***décidable****.*

References

[1] Nick Benton, John Hughes, and Eugenio Moggi. Monads and effects. In Gilles Barthe, Peter Dybjer, Luís Pinto, and João Saraiva, editors, *Applied Semantics, International Summer School, APPSEM 2000, Caminha, Portugal, September 9-15, 2000, Advanced Lectures*, volume 2395 of *Lecture Notes in Computer Science*, pages 42–122. Springer, 2000.

[2] Nick Benton and Andrew Kennedy. Exceptional syntax journal of functional programming. *J. Funct. Program.*, 11(4):395–410, 2001.

[3] Martin Hyland, Gordon D. Plotkin, and John Power. Combining effects: Sum and tensor. *Theor. Comput. Sci.*, 357(1-3):70–99, 2006.

[4] Paul Blain Levy. Call-by-push-value: Decomposing call-by-value and call-by-name. *High. Order Symb. Comput.*, 19(4):377–414, 2006.

[5] Paul Blain Levy. Monads and adjunctions for global exceptions. In Stephen D. Brookes and Michael W. Mislove, editors, *Proceedings of the 22nd Annual Conference on Mathematical Foundations of Programming Semantics, MFPS 2006, Genova, Italy, May 23-27, 2006*, volume 158 of *Electronic Notes in Theoretical Computer Science*, pages 261–287. Elsevier, 2006.

[6] Gordon D. Plotkin and Matija Pretnar. A logic for algebraic effects. In *Proceedings of the Twenty-Third Annual IEEE Symposium on Logic in Computer Science, LICS 2008, 24-27 June 2008, Pittsburgh, PA, USA*, pages 118–129. IEEE Computer Society, 2008.

Annexes

Abréviations

Afin de rendre le tout plus lisible, on va appliquer une succession d'abréviations que l'on décrit dans cette section.

Premièrement, on va simplifier les résultats binaires en résultat fini:

$$A_1 \times ... \times A_n \overset{\textbf{def}}{=} (A_1 \times ... \times A_{n-1}) \times A_n \qquad (n \geq 3)$$
$$\langle V_1, ..., V_n \rangle \overset{\textbf{def}}{=} \langle \langle V_1, ..., V_{n-1} \rangle, V_n \rangle \qquad (n \geq 3)$$

On comprend les résultats binaires, quand $n = 2$, les résultats unaires, quand $n = 1$ (simplement A_1), et le résultat vide, quand $n = 0$ ($\mathbf{1}$).

On détermine les abréviations suivantes :

$$\mathbf{f}(V_1, ..., V_n) \overset{\textbf{def}}{=} \mathbf{f}(\langle V_1, ..., V_n \rangle)$$
$$l(V_1, ..., V_n) \overset{\textbf{def}}{=} l(\langle V_1, ..., V_n \rangle)$$
$$\mathbf{op}_{V_1,...,V_n}(x : \beta.M) \overset{\textbf{def}}{=} \mathbf{op}_{\langle V_1,...,V_n \rangle}(x : \beta.M)$$
$$k(V_1, ..., V_n) \overset{\textbf{def}}{=} k(\langle V_1, ..., V_n \rangle)$$

On va omettre les parenthèses vides :

$$\mathbf{f} \overset{\textbf{def}}{=} \mathbf{f}() \qquad l \overset{\textbf{def}}{=} l() \qquad k \overset{\textbf{def}}{=} k()$$

On adapte aussi les destructeurs de tuples de taille finis. À la place d'un destructeur on va s'autoriser une écriture avec une association de multiple variables.

Exemple 12. Par exemple :

$$\mathbf{op}_{x_1,...,x_n}(k) \mapsto M \overset{\textbf{def}}{=} (\textbf{match } x \textbf{ with } \langle x_1, ..., x_n \rangle \mapsto M)$$

$$\textbf{if } V \textbf{ then } M \textbf{ else } N \overset{\textbf{def}}{=} \textbf{match } x \textbf{ with } \{\textbf{true} \mapsto M, \ \textbf{false} \mapsto N\}$$

Lorsqu'une opération ne retourne rien on va abrégé de la façon suivante :

$$\mathbf{op}_V() \overset{\textbf{def}}{=} \mathbf{op}_V(x : \mathbf{0}.\textbf{match } x \textbf{ with } \{\})$$