Typage des coroutines en logique soustractive

Tristan Crolard

UFR d'Informatique, Université Paris 7
2 Place Jussieu, 75251 Paris, France
`crolard@@ufr-info-p7.jussieu.fr`

Nous étudions le sens calculatoire de la soustraction, le connecteur dual de l'implication, en exploitant la correspondance de Curry-Howard. En nous appuyant sur la dualité et la Déduction Naturelle Classique de M. Parigot, nous définissons une restriction du $\lambda\mu$-calcul stable par réduction, dont le système de type correspond à la logique soustractive. Dans ce calcul, tout terme typable au second ordre est fortement normalisable. Les objets typés par la soustraction sont interprétés comme des contextes de coroutines de première classe.

1. Introduction

Dans cet article, nous étudions le sens calculatoire de la soustraction, le connecteur dual de l'implication, en exploitant la correspondance de Curry-Howard [14]. Rappelons brièvement cette correspondance pour la logique propositionnelle du second ordre [18]. Le vrai $\top$ correspond au type singleton, l'implication $\rightarrow$ au type des fonctions, la conjonction $\wedge$ au produit cartésien, le quantificateur universel $\forall^2$ au polymorphisme du second ordre. Le faux $\perp$ correspond au type universel, la disjonction $\vee$ à la somme disjointe, le quantificateur existentiel $\exists^2$ aux types abstraits de données [19]. Le tableau suivant récapitule ces connecteurs et quantificateurs tout en soulignant la dualité : quel est le sens calculatoire de la soustraction ?

$\top$	$\perp$
$\rightarrow$	$-$
$\wedge$	$\vee$
$\forall^2$	$\exists^2$

Remarquons aussi que la correspondance de Curry-Howard s'exprime généralement entre le λ-calcul et la déduction naturelle, systèmes dans lesquels la dualité n'est pas explicite. Il faut se placer dans la théorie de catégories bicartésiennes fermées [1] et le calcul des séquents [6] pour observer la symétrie entre la conjonction et la disjonction par exemple.

De la même manière, la symétrie entre l'implication et la soustraction n'apparaît plus dans les règles du calcul que nous proposons ici. La dualité est toutefois utilisée pour construire le calcul, et bien entendu, la logique associée au système de type sera la logique soustractive. Par exemple, le rôle essentiel de l'implication est de permettre le déchargement des hypothèses. Par conséquent, le rôle essentiel de la soustraction sera de permettre le déchargement des conclusions: le $\lambda\mu$-calcul de M. Parigot et son système de type, la Déduction Naturelle Classique (CND) sont donc bien adaptés à cette étude (dans CND les séquents ont plusieurs hypothèses et plusieurs conclusions).

D'autre part, puisque l'implication est définissable en logique classique par $A \Rightarrow B \equiv \neg A \vee B$, la soustraction est, par dualité, aussi définissable par $A - B \equiv A \wedge \neg B$. Cette définition nous donne une idée du sens calculatoire de la soustraction en logique classique : un objet de type $A - B$ peut être codé par un couple formé d'un terme (de type A) et d'une continuation réifiée en objet de première classe (de type $\neg B$). Pour plus de détails sur le lien entre le typage des opérateurs de contrôle et le sens calculatoire des preuves en logique classique, nous renvoyons à la littérature : T. G. Griffin [12], C. R. Murthy [21, 22], F. Barbanera and S. Berardi [2, 3], N. J. Rehof and M. H. Sørensen [30], P. De Groote [9], J.-L. Krivine [17]...

Afin de comprendre le sens calculatoire de la soustraction en logique soustractive (cf. section 1.1), il faut commencer par considérer une restriction de CND où la soustraction n'est pas définissable, mais où les séquents ont toujours plusieurs conclusions. Une telle restriction a été définie dans [7]. Du point de vue calculatoire, dans le $\lambda\mu$-calcul restreint, les continuations ne peuvent plus être réifiée en objet de première classe. Pour interpréter ce $\lambda\mu$-calcul restreint, il faut savoir que les opérateurs de contrôle permettent, entre autres, de simuler un mécanisme de coroutines coopérantes dans le style du langage Simula. Une implantation en Scheme, utilisant le **call/cc**, peut être trouvée dans [10, 11]. Cette approche a été étendue dans le Standard ML du New Jersey (SML/NJ) pour fournir une implantation simple et élégante des processus légers (*threads*). Remarquez que dans les implantations les plus simples, c'est le processus qui demande explicitement à être suspendu [31, 32, 34, 27, 4]. Pour des implantations complète du temps partagé (comprenant donc un ordonnanceur préemptif), cf. [4, 33].

Ce qu'il est important de noter ici, c'est que des opérateurs de contrôle comme le **call/cc** de Scheme et sa variante typée **callcc** (et **throw**) de SML/NJ permettent d'échanger les contextes de coroutines (ici le contexte d'une coroutine est exactement sa continuation). Néanmoins, dans la plupart des implantations de libraires de *threads* utilisateurs, chaque *thread* possède sa propre pile d'exécution. En effet, au cours de l'exécution d'un appel de fonction, l'environnement créé lors du passage des paramètres est local non seulement à la fonction, mais aussi au *thread*. Si l'ordonnanceur change de *thread* actif pendant l'exécution du corps de la fonction, le nouveau *thread* actif n'a évidemment plus accès à l'environnement précédent.

Dans le $\lambda\mu^+$-calcul restreint, les continuations ne sont plus des objets de première classe, mais il est toujours possible de changer de contexte. Toutefois, un contexte est maintenant un couple *environnement* + *continuation* (pour éviter toute confusion, nous appellerons μ-contexte un tel contexte). Un μ-contexte est donc exactement ce que l'on est en droit d'attendre comme contexte d'une coroutine. En résumé, dans cet article, nous considérons des μ-contextes de première classe (et donc des coroutines de première classe), typés par la soustraction.

1.1. La logique soustractive

La logique soustractive est une extension conservative de la logique intuitionniste dans le cadre propositionnel. Au premier et au second ordre, la logique soustractive est conservative sur la théorie obtenue en ajoutant à la logique intuitionniste du premier ordre (resp. du second ordre) le schéma d'axiomes DIS (resp. DIS et DIS$^2 x$ (resp. X) n'apparaît pas dans B:

$$\forall x(A \vee B) \vdash \forall x A \vee B \qquad\qquad \forall X(A \vee B) \vdash \forall X A \vee B$$

Nous renvoyons à [28, 29, 6] pour plus de détails sur la logique soustractive.

1.2. Travaux connexes

H. Nakano, Y. Kameyama et M. Sato [24, 23, 25, 15, 16, 35] ont proposé différents systèmes logiques dans le but de typer une variante (lexicale et non-confluente) du mécanisme de **catch** et **throw** issu de Lisp. Ces auteurs considèrent aussi une restriction intuitionniste de leurs calculs (dont ils ne donnent néanmoins pas d'interprétation). Cette restriction les a amené à ajouter à leurs systèmes une une nouvelle abstraction (appelée *tag-abstraction*) et qui est typée par la disjonction. Nous avons montré dans [5] que le $\lambda\mu$-calcul peut lui aussi être vu comme un λ-calcul pourvu d'un mécanisme (lexical et confluent) de **catch** et **throw**. La règle d'introduction gauche de la soustraction peut être vue comme un alternative à la *tag-abstraction* dans la mesure où cette règle décharge aussi une conclusion.

1.3. Plan de l'article

Dans la section 2, nous dérivons diverses règles (classiques) pour la soustraction en appliquant la dualité. Ce sont ces règles que nous allons contraindre pour définir la déduction naturelle correspondant à la logique soustractive. Une première restriction simple, mais qui n'est pas stable par réduction, est tout d'abord rappelée.

Dans la section 3, on rappelle la définition $\lambda\mu$-calcul avec somme disjointe et produit cartésien, noté $\lambda\mu^{+\times}$-calcul, ainsi que les règles de typage. On profite alors de la définissabilité de la soustraction en logique classique pour dériver les « macros » destinée à décorer les règles d'introduction et d'élimination ainsi que la règle de calcul (on appelle $\lambda\mu^{+\times-}$-calcul le calcul résultant). On définit alors les notions de variable visible par une coroutine et de $\lambda\mu^{+\times-}$-terme qui respecte les règles de visibilité. On prouve enfin la stabilité par réduction de l'ensemble des $\lambda\mu^{+\times-}$-termes qui respectent les règles de visibilité (et donc aussi la stabilité par réduction des preuves restreintes à la logique soustractive).

2. La logique soustractive

Nous avons vu dans l'introduction que la soustraction est définissable en logique classique par $A - B \equiv A \wedge \neg B$, par exemple. Bien entendu, il est aussi possible de définir directement ce connecteur. Nous allons dériver ces règles en exploitant la dualité et pour cela nous rappelons tout d'abord les règles de l'implication en déduction naturelle classique:

Règle d'introduction de $\Rightarrow$

$$\frac{\Gamma, A \vdash \Delta, B}{\Gamma \vdash \Delta, A \Rightarrow B}$$

Règle d'élimination de $\Rightarrow$

$$\frac{\Gamma \vdash \Delta,\ A \Rightarrow B \qquad \Gamma' \vdash \Delta',\ A}{\Gamma, \Gamma' \vdash \Delta, \Delta', B}$$

Les règles de la soustraction s'obtiennent par symétrie :

Règle d'intro. gauche du $-$

$$\frac{\Gamma, A \vdash \Delta, B}{\Gamma, A - B \vdash \Delta}$$

Règle d'élim. gauche du $-$

$$\frac{\Gamma, A - B \vdash \Delta \qquad \Gamma', B \vdash \Delta'}{\Gamma, \Gamma', A \vdash \Delta, \Delta'}$$

Notation. On appelle CND_{sym} la déduction naturelle classique étendue par les règles ci-dessus.

2.0.1. La déduction naturelle classique soustractive SND

Les règles obtenues par symétrie sont des règles d'introduction et d'élimination gauche, qui ne sont pas dans le style de la déduction naturelle (de même que les règles duales de celles de la conjonction ne sont pas celles de la disjonction en déduction naturelle). Il existe bien entendu aussi des règles d'élimination et d'introduction droite pour la soustraction.

Définition 2.0.1 *On appelle déduction naturelle soustractive SND, le système de la déduction naturelle classique CND plus les règles d'introduction et d'élimination de la soustraction suivantes:*

$$\frac{\Gamma \vdash \Delta, A \qquad \Gamma', B \vdash \Delta'}{\Gamma, \Gamma' \vdash \Delta, \Delta', A - B} \qquad\qquad \frac{\Gamma \vdash \Delta, A - B \qquad \Gamma', A \vdash \Delta', B}{\Gamma, \Gamma' \vdash \Delta, \Delta'}$$

Remarque. Les règles de SND sont dérivables à partir de celles de CND_{sym}. En effet :

- Dérivation de la règle d'élimination droite

$$\frac{\Gamma \vdash \Delta, A - B \qquad \dfrac{\Gamma', A \vdash \Delta', B}{\Gamma', A - B \vdash \Delta'}}{\Gamma, \Gamma' \vdash \Delta, \Delta'}$$

- Dérivation de la règle d'introduction droite

$$\frac{\Gamma \vdash \Delta, A \qquad \dfrac{A - B \vdash A - B \qquad \Gamma', B \vdash \Delta'}{\Gamma', A \vdash \Delta', A - B}}{\Gamma, \Gamma' \vdash \Delta, \Delta', A - B}$$

Réciproquement, les règles de CND_{sym} sont dérivables à partir de celles de SND. En effet :

– Dérivation de la règle d'introduction gauche

$$\frac{A - B \vdash A - B \qquad \Gamma, A \vdash \Delta, B}{\Gamma, A - B \vdash \Delta}$$

– Dérivation de la règle d'élimination gauche

$$\frac{\dfrac{A \vdash A \qquad \Gamma', B \vdash \Delta'}{\Gamma' \vdash \Delta', A - B} \qquad \Gamma, A - B \vdash \Delta}{\Gamma, \Gamma', A \vdash \Delta, \Delta'}$$

2.1. Restrictions intuitionnistes

Les règles données pour l'implication sont classiques puisque la règle d'introduction permet de dériver le tiers-exclus pour la négation intuitionniste définie par $\neg A \equiv A \Rightarrow \bot$ (qui vérifie déjà la règle de la contradiction) :

$$\cfrac{\cfrac{A \vdash A}{A \vdash A, \bot}}{\vdash A, \neg A}$$

Il est connu que l'on peut retrouver la logique intuitionniste dans le cadre propositionnel en restreignant uniquement la règle d'implication à des séquents ayant au plus une conclusion. Toutefois au premier ordre (resp. au second ordre) il est aussi possible de dériver DIS (resp. DIS^2) dès que la virgule de droite dénote une disjonction (cf. [7]). Voici la preuve de DIS :

$$\cfrac{\cfrac{\cfrac{\cfrac{A \vee B \vdash A \vee B}{\forall x(A \vee B) \vdash A \vee B}}{\forall x(A \vee B) \vdash A, B}}{\forall x(A \vee B) \vdash \forall x A, B}}{\forall x(A \vee B) \vdash \forall x A \vee B}$$

Les règles de la soustraction sont aussi classiques puisque la règle d'introduction gauche permet de dériver la règle de la contradiction pour la négation faible définie par $\sim\! A \equiv \bot - A$ (qui vérifie déjà la règle du tiers exclus) :

$$\cfrac{\cfrac{A \vdash A}{A, \top \vdash A}}{A, \sim\! A \vdash}$$

Toujours par dualité, on sait qu'il suffit de restreindre la règle d'introduction gauche à au plus une hyphothèse pour rester intuitionniste dans le cadre propositionnel, et conservatif sur la théorie contenant uniquement le schéma DIS au premier et au second ordre [6]. Remarquez que l'axiome dual de DIS (i.e. l'axiome $\exists x A \wedge B \vdash \exists x(A \wedge B)$) était déjà dérivable en logique intuitionniste.

Toutefois la restriction des séquents à une conclusion (resp. une hypothèse) n'est pas stable par réduction: par la correspondance de Curry-Howard cette restriction correspond à supposer certains sous-termes μ-clos (ne contenant aucune μ-variable libre) et la μ-clôture n'est pas une propriété stable par β-réduction. Il est toutefois possible de définir une contrainte plus faible mais définissant la même logique et qui soit stable par réduction (cf. [8, 7]).

Nous allons maintenant définir cette contrainte, mais en considérant maintenant les $\lambda\mu$-termes correspondant aux preuves (par l'isomorphisme de Curry-Howard). Nous prouvons alors que cette contrainte est stable par réduction.

3. Le $\lambda\mu^{+\times-}$-calcul typé

Nous définissons tout d'abord dans cette section le $\lambda\mu^{+\times-}$-calcul, dont le système de type est SND. Nous définissons ensuite une notion de variable utile à une coroutine, puis des règles de visibilité qui restreignent le système de type à la logique soustractive.

3.1. Le $\lambda\mu^{+\times}$-calcul pur

Rappelons tout d'abord la syntaxe des $\lambda\mu^{+\times}$-termes (comme d'habitude, on note $x, y, z, \ldots$ les λ-variables et $\alpha, \beta, \gamma, \ldots$ les μ-variables).

Définition 3.1.1 *Si t, u, v sont des $\lambda\mu^{+\times}$-termes alors :*

$$\begin{array}{l} x, \quad (u\ v), \quad \lambda x.t, \quad \mu\alpha[\beta]t, \\ \langle u, v\rangle, \quad \mathbf{match}\ t\ \mathbf{with}\ \langle x, y\rangle \mapsto u, \\ \mathbf{inl}\ t, \quad \mathbf{inr}\ t, \quad \mathbf{cases}\ t\ \mathbf{of}\ (\mathbf{inl}\ x) \mapsto u \mid (\mathbf{inr}\ y) \mapsto v \end{array}$$

sont aussi des $\lambda\mu^{+\times}$-termes.

Règle de réduction du $\lambda\mu^{+\times}$-calcul

- $(\lambda x.u\ v) \rightsquigarrow u\{v/x\}$
- $(\mu\alpha u\ v) \rightsquigarrow \mu\alpha u\{[\alpha](t\ v)/[\alpha]t\}$

- $[\beta]\mu\alpha t \leadsto t\{\beta/\alpha\}$
- $\mu\alpha[\alpha]t \leadsto t$ si α n'apparaît pas libre dans t.
- $\textbf{match}\ \langle u, v\rangle\ \textbf{with}\ \langle x, y\rangle \mapsto t \leadsto t\{u/x, v/y\}$
- $\textbf{cases}\ (\textbf{inl}\ t)\ \textbf{of}\ (\textbf{inl}\ x) \mapsto u \mid (\textbf{inr}\ y) \mapsto v \leadsto u\{t/x\}$
- $\textbf{cases}\ (\textbf{inr}\ t)\ \textbf{of}\ (\textbf{inl}\ x) \mapsto u \mid (\textbf{inr}\ y) \mapsto v \leadsto v\{t/y\}$
- $\textbf{let}\ x = u\ \textbf{in}\ t \leadsto t\{u/x\}$

La notation $u\{v/x\}$ correspond à la substitution habituelle de la λ-variable x par v dans u. La notation $u\{[\alpha](t\ v)/[\alpha]t\}$ signifie « remplacer dans le terme u toute occurrence d'un sous-terme de la forme $[\alpha]t$ par $[\alpha](t\ v)$ ».

3.2. Le $\lambda\mu^{+\times}$-calcul typé

Axiome

$$x : A^x \vdash A$$

Règles de coupure

$$\frac{u : \Gamma \vdash \Delta; A \qquad t : \Gamma, A^x \vdash \Delta; B}{\textbf{let}\ x = u\ \textbf{in}\ t : \Gamma \vdash \Delta; B}$$

Règle d'affaiblissement gauche

$$\frac{t : \Gamma \vdash \Delta; B}{t : \Gamma, A^x \vdash \Delta; B}$$

Règle de contraction gauche

$$\frac{t : \Gamma, A^x, A^y \vdash \Delta; B}{t\{z/x, z/y\} : \Gamma, A^z \vdash \Delta; B}$$

Règles du $\wedge$

$$\frac{u : \Gamma \vdash \Delta; A \qquad v : \Gamma \vdash \Delta; B}{\langle u, v\rangle : \Gamma \vdash \Delta; A \wedge B}$$

$$\frac{t : \Gamma \vdash \Delta; A \wedge B \qquad u : \Gamma, A^x, B^y \vdash \Delta; C}{\textbf{match } t \textbf{ with } \langle x, y\rangle \mapsto u : \Gamma \vdash \Delta; C}$$

Règles du $\vee$

$$\frac{t : \Gamma \vdash \Delta; A}{\textbf{inl } t : \Gamma \vdash \Delta; A \vee B} \qquad \frac{t : \Gamma \vdash \Delta; B}{\textbf{inr } t : \Gamma \vdash \Delta; A \vee B}$$

$$\frac{t : \Gamma \vdash \Delta; A \vee B \qquad u : \Gamma, A^x \vdash \Delta; C \qquad v : \Gamma, B^y \vdash \Delta; C}{\textbf{cases } t \textbf{ of } (\textbf{inl } x) \mapsto u \mid (\textbf{inr } y) \mapsto v : \Gamma \vdash \Delta; C}$$

Règles du $\rightarrow$

$$\frac{t : \Gamma, A^x \vdash \Delta; B}{\lambda x.t : \Gamma \vdash \Delta; A \rightarrow B} \qquad \frac{u : \Gamma \vdash \Delta; A \rightarrow B \qquad v : \Gamma \vdash \Delta; A}{(u\ v) : \Gamma \vdash \Delta; B}$$

3.2.1. Règles des quantificateurs

Comme dans le système AF_2 de J.-L. Krivine [18] les règles d'introduction et d'élimination des quantificateurs ne sont pas explicitée par les $\lambda\mu$-termes qui décorent les règles. les règles des connecteurs $\exists, \exists^2$ peuvent d'obtenir soit par dualité, soit à partir de leur définition au second ordre.

Règles du $\forall$

$$\frac{u : \Gamma \vdash \Delta; A}{u : \Gamma \vdash \Delta; \forall x A} \qquad \frac{u : \Gamma \vdash \Delta; \forall x A}{u : \Gamma \vdash \Delta; A\{t/x\}}$$

Règles du $\forall^2$

$$\frac{u : \Gamma \vdash \Delta; A}{u : \Gamma \vdash \Delta; \forall X A} \qquad \frac{u : \Gamma \vdash \Delta; \forall X A}{u : \Gamma \vdash \Delta; A\{T/X\}}$$

3.2.2. Règles de nommage

Ces règles sont « les règles » du $\lambda\mu$-calcul, puisqu'elles permettent de gérer les conclusions multiples. Remarquons qu'un terme « nommé » (*i.e.* de la forme $[\alpha]t$) décore un séquent ne contenant aucune formule non nommée, elle est donc nécessairement suivie d'une règle μ. Cette contrainte correspond à la contraint syntaxique des $\lambda\mu$-termes où toute occurrence du constructeur μ est nécessairement de la forme $\mu\alpha[\beta]$.

$$\frac{t : \Gamma \vdash \Delta; A}{[\alpha]t : \Gamma \vdash \Delta, A^\alpha;} \qquad \frac{t : \Gamma \vdash \Delta, A^\alpha;}{\mu\alpha.t : \Gamma \vdash \Delta; A}$$

Remarque. Dans la règle μ, la formule A^α peut ne pas figurer dans le séquent hypothèse : dans ce cas le sens logique de cette règle est l'affaiblissement à droite. Dans la règle [ ], la formule A^α peut déjà figurer dans le Δ du séquent hypothèse : dans ce cas, le sens logique de cette règle est la contraction à droite. Nous allons maintenant isoler ces cas particuliers pour obtenir les règles dérivées d'affaiblissement et de contraction à droite.

- Règle d'affaiblissement droite (où B^β n'apparaît pas dans Δ)

$$\frac{\dfrac{t : \Gamma \vdash \Delta; A}{[\alpha]t : \Gamma \vdash \Delta, A^\alpha;}}{\mu\beta[\alpha]t : \Gamma \vdash \Delta, A^\alpha; B}$$

- Règle de contraction droite

$$\frac{\dfrac{t : \Gamma \vdash \Delta, A^\alpha; t : A}{[\alpha]t : \Gamma \vdash \Delta, A^\alpha;}}{\mu\alpha[\alpha]t : \Gamma \vdash \Delta; A}$$

Ces deux cas particuliers (ces deux « macros ») ont un comportement proche des instructions **catch** et **throw** de certains langages Lisp (cf. [7]). Nous utiliserons aussi la terminologie **get-context** et **set-context** (inspirée de la *Single Unix Specification* [13]) qui sera mieux adaptée à la restriction de ces opérateurs que nous allons considérer dans la suite. Voici les règles de typage dérivées pour ces opérateurs.

3.2.3. Typage des opérateurs set-context et get-context

Règle de contraction droite

$$\frac{t : \Gamma \vdash \Delta, A^{\alpha}; A}{\textbf{get-context}\ \alpha\ t : \Gamma \vdash \Delta; A}$$

Règle d'affaiblissement droite

$$\frac{t : \Gamma \vdash \Delta; A}{\textbf{set-context}\ \alpha\ t : \Gamma \vdash \Delta, A^{\alpha}; B}$$

3.3. Continuations de première classe

Les μ-variables (qui dénotent des continuations) ne sont pas des objets de première classe dans le $\lambda\mu$-calcul. En revanche, une continuation α peut être réifiée en l'objet de première classe $\lambda x.$**throw** $\alpha\ x$. Toutefois cette forme n'est pas stable pas réduction pour les règles du $\lambda\mu$-calcul. La forme générale (stable par réduction) d'une continuation de première classe dans le $\lambda\mu$-calcul est en fait $\lambda x.$**throw** $\alpha\ \mathcal{C}[x]$, où $\mathcal{C}[\bullet]$ est un contexte. Rappelons tout qu'un contexte est un terme qui contient un « trou », noté $\bullet$, que l'on peut voir comme un symbole de constante (ou une variable libre) et qui apparaît exactement une fois dans le terme. Les contextes que nous allons considérer ici on tous la forme suivante $(\bullet\ t_1\ \ldots\ t_n)$ où $t_1, \ldots, t_n$ sont des termes. Cette forme est due aux règles de réduction choisies dans le $\lambda\mu$-calcul, et n'aura pas de conséquences pour la suite: nous garderons donc plutôt la notation générique $\mathcal{C}[\bullet]$. Ces contextes peuvent être vus comme les continuations partielles [20] qui ont déjà été transmise au **throw** par le **catch**. Au cours de l'évaluation, le nom α peut changer, ainsi que la continuation partielle $\mathcal{C}[\bullet]$, mais la forme reste la même.

3.4. Le $\lambda\mu^{+\times-}$-calcul

Dans cette section on utilise le fait que la soustraction est définissable en logique classique (par $A - B \equiv A \wedge \neg B$) pour dériver la normalisation forte au second ordre de notre calcul ainsi que l'unicité de la forme normale. Pour cela, on plonge notre calcul dans le $\lambda\mu$-calcul de la façon suivante:

– **pack-context** $t\ \alpha\ \mathcal{C}[\bullet] \equiv \langle t, \lambda x.$**set-context** $\alpha\ \mathcal{C}[x]\rangle$

– **resume** c **with** $x \mapsto u \equiv$ **match** c **with** $\langle x, k\rangle \mapsto (k\ u)$

Nous décorons ici les règles d'introduction et d'élimination de la soustraction. On restreint pour cela la règle d'introduction: la preuve du séquent de droite doit avoir la forme d'un contexte. Remarquons que ce n'est pas une restriction du point de vue de la prouvabilité: le contexte vide suffirait (c'est-à-dire l'axiome $B \vdash B$, il suffit de faire suivre cette règle d'une coupure pour retrouver la règle générale).

Règle d'introduction

$$\frac{t : \Gamma \vdash \Delta; A \qquad \mathcal{C}[\bullet] : \Gamma, B^{\bullet} \vdash \Delta; C}{\textbf{pack-context}\ t\ \gamma\ \mathcal{C}[\bullet] : \Gamma \vdash \Delta, C^{\gamma}; A - B}$$

Règle d'élimination

$$\frac{c : \Gamma \vdash \Delta; A - B \qquad u : \Gamma, A^{x} \vdash \Delta; B}{\textbf{resume}\ c\ \textbf{with}\ x \mapsto u : \Gamma \vdash \Delta; C}$$

Règle de calcul dérivée

$$\begin{array}{c}\textbf{resume}\ (\textbf{pack-context}\ t\ \gamma\ \mathcal{C}[\bullet])\ \textbf{with}\ x \mapsto u \\ \leadsto \textbf{set-context}\ \gamma\ \mathcal{C}[u\{t/x\}]\end{array}$$

Remarque. Le $\lambda\mu^{+\times -}$-calcul n'est pour l'instant rien d'autre que le $\lambda\mu^{+\times}$-calcul (on a seulement défini des macros). Il est par conséquent toujours fortement normalisant au second ordre [26].

3.4.1. Notion de variable utile à une coroutine

Pour chaque $\lambda\mu$-terme t, on définit l'ensemble $\mathcal{S}_{[]}$ des λ-variables utiles à la routine courante (i.e. les λ-variables libres qui apparaissent dans le μ-contexte courant) et, pour tout μ-variable δ libre dans t, l'ensemble $\mathcal{S}_{\delta}(t)$ des λ-variables utiles à la coroutine δ (i.e. les λ-variables libres qui apparaissent dans le μ-contexte δ). Cette définition s'applique aussi aux contextes.

Définition 3.4.1 *On définit par récurrence simultanée sur t les* *ensembles* $\mathcal{S}_{\Box}(t)$ *et* $\mathcal{S}_\delta(t)$ *pour toute* μ*-variable* δ *libre dans* t*:*

- $\mathcal{S}_{\Box}(\bullet) = \emptyset$
  $\mathcal{S}_\delta(\bullet) = \emptyset$
- $\mathcal{S}_{\Box}(x) = \{x\}$
  $\mathcal{S}_\delta(x) = \emptyset$
- $\mathcal{S}_{\Box}(\lambda x.u) = \mathcal{S}_{\Box}(u)\backslash\{x\}$
  $\mathcal{S}_\delta(\lambda x.u) = \mathcal{S}_\delta(u)\backslash\{x\}$
- $\mathcal{S}_{\Box}(u\ v) = \mathcal{S}_{\Box}(u) \cup \mathcal{S}_{\Box}(v)$
  $\mathcal{S}_\delta(u\ v) = \mathcal{S}_\delta(u) \cup \mathcal{S}_\delta(v)$
- $\mathcal{S}_{\Box}([\alpha]u) = \emptyset$
  $\mathcal{S}_\alpha([\alpha]u) = \mathcal{S}_\alpha(u) \cup \mathcal{S}_{\Box}(u)$
  $\mathcal{S}_\delta([\alpha]u) = \mathcal{S}_\delta(u)$ *pour tout* $\delta \neq \alpha$
- $\mathcal{S}_{\Box}(\mu\alpha.u) = \mathcal{S}_\alpha(u)$
  $\mathcal{S}_\delta(\mu\alpha.u) = \mathcal{S}_\delta(u)$
- $\mathcal{S}_{\Box}(\langle u, v\rangle) = \mathcal{S}_{\Box}(u) \cup \mathcal{S}_{\Box}(v)$
  $\mathcal{S}_\delta(\langle u, v\rangle) = \mathcal{S}_\delta(u) \cup \mathcal{S}_\delta(v)$
- $\mathcal{S}_{\Box}(\textbf{match}\ p\ \textbf{with}\ \langle x, y\rangle \mapsto t) = \mathcal{S}_{\Box}(t)[\mathcal{S}_{\Box}(p)/x, \mathcal{S}_{\Box}(p)/y]$
  $\mathcal{S}_\delta(\textbf{match}\ p\ \textbf{with}\ \langle x, y\rangle \mapsto t) = \mathcal{S}_\delta(t)[\mathcal{S}_{\Box}(p)/x, \mathcal{S}_{\Box}(p)/y] \cup \mathcal{S}_\delta(p)$
- $\mathcal{S}_{\Box}(\textbf{inl}\ u) = \mathcal{S}_{\Box}(u)$ *et* $\mathcal{S}_{\Box}(\textbf{inr}\ u) = \mathcal{S}_{\Box}(u)$
  $\mathcal{S}_\delta(\textbf{inl}\ u) = \mathcal{S}_\delta(u)$ *et* $\mathcal{S}_\delta(\textbf{inr}\ u) = \mathcal{S}_\delta(u)$
- $\mathcal{S}_{\Box}(\textbf{cases}\ w\ \textbf{of}\ (\textbf{inl}\ x) \mapsto u \mid (\textbf{inr}\ y) \mapsto v) =$
  $\quad \mathcal{S}_{\Box}(u)[\mathcal{S}_{\Box}(w)/x] \cup \mathcal{S}_{\Box}(v)[\mathcal{S}_{\Box}(w)/y]$
  $\mathcal{S}_\delta(\textbf{cases}\ w\ \textbf{of}\ (\textbf{inl}\ x) \mapsto u \mid (\textbf{inr}\ y) \mapsto v) =$
  $\quad \mathcal{S}_\delta(u)[\mathcal{S}_{\Box}(w)/x] \cup \mathcal{S}_\delta(v)[\mathcal{S}_{\Box}(w)/y] \cup \mathcal{S}_\delta(w)$
- $\mathcal{S}_{\Box}(\textbf{let}\ x = v\ \textbf{in}\ u) = \mathcal{S}_{\Box}(u)[\mathcal{S}_{\Box}(v)/x]$
  $\mathcal{S}_\delta(\textbf{let}\ x = v\ \textbf{in}\ u) = \mathcal{S}_\delta(u)[\mathcal{S}_{\Box}(v)/x] \cup \mathcal{S}_\delta(v)$
- $\mathcal{S}_{\Box}(\textbf{pack-context}\ t\ \alpha\ \mathcal{C}[\bullet]) = \mathcal{S}_{\Box}(t)$
  $\mathcal{S}_\alpha(\textbf{pack-context}\ t\ \alpha\ \mathcal{C}[\bullet]) = \mathcal{S}_\alpha(t) \cup \mathcal{S}_\alpha(\mathcal{C}[\bullet]) \cup \mathcal{S}_{\Box}(t) \cup \mathcal{S}_{\Box}(\mathcal{C}[\bullet])$
  $\mathcal{S}_\delta(\textbf{pack-context}\ t\ \alpha\ \mathcal{C}[\bullet]) = \mathcal{S}_\delta(t) \cup \mathcal{S}_\delta(\mathcal{C}[\bullet])$ *pour tout* $\delta \neq \alpha$
- $\mathcal{S}_{\Box}(\textbf{resume}\ c\ \textbf{with}\ x \mapsto u) = \mathcal{S}_{\Box}(u)\backslash\{x\} \cup \mathcal{S}_{\Box}(c)$
  $\mathcal{S}_\delta(\textbf{resume}\ c\ \textbf{with}\ x \mapsto u) = \mathcal{S}_\delta(u)[\mathcal{S}_{\Box}(c)/x] \cup \mathcal{S}_\delta(c)$

Remarque. Étant donné un $\lambda\mu$-terme t, une λ-variable libre de t peut être utile à plusieurs coroutines dans t en même temps (y compris éventuellement la routine courante).

Remarque. Dans le cas particulier des abbréviations **set-context** et **get-context**, la définition précédente nous donne:

- Si le $\lambda\mu$**ct**-terme est **get-context** $\alpha\ u$ (i.e. $\mu\alpha[\alpha]u$) alors :

$\mathcal{S}_{[]}(\mu\alpha[\alpha]u) = \mathcal{S}_\alpha([\alpha]u) = \mathcal{S}_{[]}(u) \cup \mathcal{S}_\alpha(u)$
$\mathcal{S}_\delta(\mu\alpha[\alpha]u) = \mathcal{S}_\delta([\alpha]u) = \mathcal{S}_\delta(u)$

- Si le $\lambda\mu$**ct**-terme est **set-context** α u (i.e. $\mu\beta[\alpha]u$ où β n'apparaît pas dans $[\alpha]u$) alors :
  $\mathcal{S}_{[]}(\mu\beta[\alpha]u) = \mathcal{S}_\beta([\alpha]u) = \emptyset$
  $\mathcal{S}_\alpha(\mu\beta[\alpha]u) = \mathcal{S}_\alpha([\alpha]u) = \mathcal{S}_\alpha(u) \cup \mathcal{S}_{[]}(u)$
  $\mathcal{S}_\delta(\mu\beta[\alpha]u) = \mathcal{S}_\delta([\alpha]u) = \mathcal{S}_\delta(u)$ for any $\delta \neq \alpha$

3.4.2. Règles de visibilité

La définition précédente nous permet de formaliser la notion de « coroutine qui n'accède pas aux variables locales d'une autre coroutine » en définissant l'ensemble des termes qui respectent les règles de visibilité des coroutines. En bref, un terme respecte les règles de visibilité des coroutines si toute variable utile est visible.

Définition 3.4.2 *On dit qu'un* $\lambda\mu^{+\times-}$*-terme* t respecte les règles de visibilité des coroutines *si et seulement si:*

1. *pour tout sous-terme de* t *de la forme* $\lambda x.u$*, pour toute* μ*-variable* δ *libre dans* u*,* $x \notin \mathcal{S}_\delta(u)$ *;*
2. *pour tout sous-terme of* t *de la forme* **resume** c **with** $x \mapsto u$*,* $\mathcal{S}_{[]}(u) \subseteq \{x\}$*.*

Remarque.

- Ne sont visibles dans une coroutine que les λ-variables déjà déclarées avant sa propre déclaration (par l'instruction **get-context**), autrement dit son environnement de déclaration. En effet, dans $\lambda x.u$, x n'est visible que de la routine courante et ne doit donc pas être utile à une autre coroutine dans u.
- Si la coroutine a été réifiée en objet de première classe, l'environnement de déclaration n'est plus connu, par conséquent on doit se restreindre à la variable x, qui sera liée lors de l'exécution au terme donné comme premier argument de **pack-context**. Remarquez que l'environnement de déclaration de la coroutine est encore connu au moment où elle est réifiée.

3.4.3. Stabilité par réduction

Dans cette section, nous montrons que le sous-ensemble des $\lambda\mu^{+\times-}$-termes qui *respectent les règles de visibilité des coroutines* est stable par réduction pour les règles du $\lambda\mu^{+\times-}$-calcul. On a déjà montré dans [7] que le sous-ensemble des $\lambda\mu^+$-termes est stable par réduction pour les règles du $\lambda\mu^+$-calcul. Ce résultat s'étend directement au $\lambda\mu^{+\times}$-calcul puisque les règles du produit que nous considérons sont dérivables au second ordre (avec la même notion de λ-variable utile, ce qui n'est pas le cas pour la somme disjointe, cf. [7]).

Lemme 3.4.3 *$\mathcal{S}_\delta(t) = \emptyset$ si δ n'apparaît pas dans t.*

Lemme 3.4.4 *Si u et v sont des $\lambda\mu$-termes et x est λ-variable libre qui apparaît dans u mais pas dans v alors pour toute μ-variable δ libre dans $u\{v/x\}$:*

$$\mathcal{S}_{[]}(u\{v/x\}) = \mathcal{S}_{[]}(u)[\mathcal{S}_{[]}(v)/x]$$
$$\mathcal{S}_\delta(u\{v/x\}) = \mathcal{S}_\delta(u)[\mathcal{S}_{[]}(v)/x] \cup \mathcal{S}_\delta(v)$$

Lemme 3.4.5 *Si u est un $\lambda\mu$-terme, $\mathcal{C}[\bullet]$ un contexte, et x est λ-variable libre qui apparaît dans u alors pour toute μ-variable δ libre dans $\mathcal{C}[u]$:*

$$\mathcal{S}_{[]}(\mathcal{C}[u]) = \mathcal{S}_{[]}(\mathcal{C}[\bullet]) \cup \mathcal{S}_{[]}(u)$$
$$\mathcal{S}_\delta(\mathcal{C}[u]) = \mathcal{S}_\delta(\mathcal{C}[\bullet]) \cup \mathcal{S}_\delta(u)$$

Lemme 3.4.6 *Étant donné une instance $r \rightsquigarrow s$ d'une règle du $\lambda\mu^{+\times-}$-calcul, si r respecte les règles de visibilité des coroutines alors $\mathcal{S}_{[]}(s) \subset \mathcal{S}_{[]}(r)$ et $\mathcal{S}_\delta(s) \subset \mathcal{S}_\delta(r)$ pour toute μ-variable δ libre dans s.*

Preuve. Les règles du $\lambda\mu^+$-calcul ont déjà été traitée dans [7]. Les règles du produit cartésien $\times$ étant dérivables, il reste donc à considérer une instance $r \rightsquigarrow s$ de la règle:

$$\begin{array}{l}\textbf{resume}\ (\textbf{pack-context}\ v\ \beta\ \mathcal{C}[\bullet])\ \textbf{with}\ x \mapsto u \\ \qquad\qquad \rightsquigarrow \textbf{set-context}\ \beta\ \mathcal{C}[u\{v/x\}]\end{array}$$

Soit y une λ-variable libre de s (et donc une λ-variable libre de r) et soit δ une μ-variable δ libre dans s (et donc une μ-variable δ libre dans r). On a:

- $\mathcal{S}_{[]}(\textbf{set-context}\ \beta\ \mathcal{C}[u\{v/x\}]) = \emptyset \subseteq \mathcal{S}_{[]}(\textbf{resume}\ (\textbf{pack-context}\ v\ \beta\ \mathcal{C}[\bullet])\ \textbf{with}\ x \mapsto u)$
- $\mathcal{S}_\beta(\textbf{set-context}\ \beta\ \mathcal{C}[u\{v/x\}]) = \mathcal{S}_\beta(\mathcal{C}[u\{v/x\}]) \cup \mathcal{S}_{[]}(\mathcal{C}[u\{v/x\}])$
  $= \mathcal{S}_\beta(\mathcal{C}[\bullet]) \cup \mathcal{S}_\beta(u\{v/x\}) \cup \mathcal{S}_{[]}(\mathcal{C}[\bullet]) \cup \mathcal{S}_{[]}(u\{v/x\})$
  $\subseteq \mathcal{S}_\beta(\mathcal{C}[\bullet]) \cup \mathcal{S}_\beta(u)[\mathcal{S}_{[]}(v)/x] \cup \mathcal{S}_\beta(v) \cup \mathcal{S}_{[]}(\mathcal{C}[\bullet]) \cup \mathcal{S}_{[]}(u)[\mathcal{S}_{[]}(v)/x]$
  $\subseteq \mathcal{S}_\beta(\mathcal{C}[\bullet]) \cup \mathcal{S}_\beta(u)[\mathcal{S}_{[]}(v)/x] \cup \mathcal{S}_\beta(v) \cup \mathcal{S}_{[]}(\mathcal{C}[\bullet]) \cup \mathcal{S}_{[]}(v)$ car $\mathcal{S}_{[]}(u) \subseteq \{x\}$
  $= \mathcal{S}_\beta(u)[\mathcal{S}_{[]}(\mathbf{v})/x] \cup \big(\mathcal{S}_\beta(v) \cup \mathcal{S}_\beta(\mathcal{C}[\bullet]) \cup \mathcal{S}_{[]}(v) \cup \mathcal{S}_{[]}(\mathcal{C}[\bullet])\big)$

$= \mathcal{S}_\beta(u)[\mathcal{S}_{[]}(\mathbf{v})/x] \cup \mathcal{S}_\beta(\textbf{pack-context } v\ \beta\ \mathcal{C}[\bullet])$
$= \mathcal{S}_\beta(u)[\mathcal{S}_{[]}(\textbf{pack-context } v\ \beta\ \mathcal{C}[\bullet])/x] \cup \mathcal{S}_\beta(\textbf{pack-context } v\ \beta\ \mathcal{C}[\bullet])$
$= \mathcal{S}_\beta(\textbf{resume } (\textbf{pack-context } v\ \beta\ \mathcal{C}[\bullet]) \textbf{ with } x \mapsto u)$

– $\mathcal{S}_\delta(\textbf{set-context } \beta\ \mathcal{C}[u\{v/x\}]) = \mathcal{S}_\delta(\mathcal{C}[u\{v/x\}])$
$= \mathcal{S}_\delta(\mathcal{C}[\bullet]) \cup \mathcal{S}_\delta(u\{v/x\})$
$\subseteq \mathcal{S}_\delta(\mathcal{C}[\bullet]) \cup \mathcal{S}_\delta(u)[\mathcal{S}_{[]}(v)/x] \cup \mathcal{S}_\delta(v)$
$= \mathcal{S}_\delta(u)[\mathcal{S}_{[]}(v)/x] \cup (\mathcal{S}_\delta(v) \cup \mathcal{S}_\delta(\mathcal{C}[\bullet]))$
$= \mathcal{S}_\delta(u)[\mathcal{S}_{[]}(\mathbf{v})/x] \cup \mathcal{S}_\delta(\textbf{pack-context } v\ \beta\ \mathcal{C}[\bullet])$
$= \mathcal{S}_\delta(u)[\mathcal{S}_{[]}(\textbf{pack-context } v\ \beta\ \mathcal{C}[\bullet])/x] \cup \mathcal{S}_\delta(\textbf{pack-context } v\ \beta\ \mathcal{C}[\bullet])$
$= \mathcal{S}_\delta(\textbf{resume } (\textbf{pack-context } v\ \beta\ \mathcal{C}[\bullet]) \textbf{ with } x \mapsto u)$

□

Lemme 3.4.7 *Étant donnés deux $\lambda\mu$-termes t, u et un contexte $C[\bullet]$, si $\mathcal{S}_{[]}(u) \subset \mathcal{S}_{[]}(t)$ et $\mathcal{S}_\delta(u) \subset \mathcal{S}_\delta(t)$ alors $\mathcal{S}_{[]}(C[u]) \subset \mathcal{S}_{[]}(C[t])$ et $\mathcal{S}_\delta(C[u]) \subset \mathcal{S}_\delta(C[t])$ pour toute μ-variable δ libre dans $C[t]$.*

Preuve. Par récurrence sur le contexte $C[\bullet]$. □

Proposition 3.4.8 *Étant donné un $\lambda\mu$-terme t, si t respecte les règles de visibilité et $t \leadsto u$ alors $\mathcal{S}_{[]}(u) \subset \mathcal{S}_{[]}(t)$ et $\mathcal{S}_\delta(u) \subset \mathcal{S}_\delta(t)$ pour toute μ-variable δ libre dans t.*

Preuve. Par le lemme 3.4.6 et le lemme 3.4.7. □

Lemme 3.4.9 *Étant donnés deux $\lambda\mu$-termes u, v, si u et v respectent les deux règles de visibilité alors $u\{v/x\}$ respecte aussi les deux règles de visibilité.*

Lemme 3.4.10 *Étant donné une instance $r \leadsto s$ d'une règle du $\lambda\mu$-calcul, si r respecte les deux règles de visibilité alors s respecte aussi les deux règles de visibilité.*

Théorème 3.4.11 *Étant donné un $\lambda\mu$-terme t, si t respecte les deux règles de visibilité et $t \leadsto u$ alors u respecte aussi les deux règles de visibilité.*

Références

[1] A. Asperti and G. Longo. *Categories, Types and Structures.* MIT Press, 1991.

[2] F. Barbanera and S. Berardi. A Symmetric Lambda Calculus for "Classical" Program Extraction. In *Theoretical Aspects of Computer Software*, volume 542 of *LNCS*, pages 495–515. Springer-Verlag, 1994.

[3] F. Barbanera and S. Berardi. Extracting constructive content from classical logic via control-like reductions. volume 662 of *LNCS*, pages 47–59. Springer-Verlag, 1994.

[4] E. C. Cooper and J. G. Morrisett. Adding threads to standard ML. Report CMU-CS-90-186, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 1990.

[5] T. Crolard. A confluent lambda-calculus with a catch/throw mechanism. Submitted to the *Journal of Functional Programming*. (ftp://sweet-smoke.ufr-info-p7.jussieu.fr/crolard/jfp.ps).

[6] T. Crolard. From bicartesian closed categories with coexponents towards subtractive logic. Submitted to *Theoretical Computer Science*.
(ftp://sweet-smoke.ufr-info-p7.jussieu.fr/crolard/tcs.ps).

[7] T. Crolard. An Intuitionistic Restriction of the $\lambda\mu$-calculus. Submitted to *Logic in Computer Science 99*.
(ftp://sweet-smoke.ufr-info-p7.jussieu.fr/crolard/lics99.ps).

[8] T. Crolard. Extension de l'isomorphisme de Curry-Howard au traitement des exceptions (application d'une étude de la dualité en logique intuitionniste). Thèse de Doctorat. Université Paris 7, 1996.

[9] P. de Groote. A simple calculus of exception handling. In *Second International Conference on Typed Lambda Calculi and Applications*, LNCS, pages 201–215, Edinburgh, United Kingdom, 1995.

[10] D. P. Friedman, C. T. Haynes, and M. Wand. Continuations and coroutines: An exercise in metaprogramming. In *Proc. 1984 ACM Symposium on Lisp and Functional Programming*, pages 293–298, August 1984.

[11] D. P. Friedman, C. T. Haynes, and M. Wand. Obtaining coroutines with continuations. *Journal of Computer Languages*, 11(3/4):143–153, 1986.

[12] T. G. Griffin. A formulæ-as-type notion of control. In *Conference Record of the Seventeenth Annual ACM Symposium on Principles of Programming Langages*, pages 47–58, 1990.

[13] The Open Group. The Single UNIX Specification, Version 2, 1997. (http://www.UNIX-systems.org/online.html).

[14] W. A. Howard. The formulæ-as-types notion of constructions. In *To H.B. Curry: Essays on Combinatory Logic, Lambda-Calculs and Formalism*, pages 479–490. Academic Press, 1987.

[15] Y. Kameyama. A new formulation of the catch/throw mechanism. In T. Ida, A. Ohori, and M. Takeichi, editors, *Second Fuji International Workshop on Functional and Logic Programming*, Word Scientific, pages 106–122, 1997.

[16] Y. Kameyama and M. Sato. A classical catch/throw calculus with tag abstraction and its strong normalizability. In X. Lin, editor, *Proc. the 4th Australasian Theory Symposium*, volume 20-3 of *Australian Computer Science Communications*, pages 183–197. Springer-Verlag, 1998.

[17] J.-L. Krivine. Classical logic, storage operators and second order λ-calculus. *Ann. of Pure and Appl. Logic*, 68:53–78, 1994.

[18] J.-L. Krivine and M. Parigot. Programming with proofs. *J. Inf. Process. Cybern. EIK*, 26(3):149–167, 1990.

[19] J. C. Mitchell and G. D. Plotkin. Abstract types have existential types. In *12th Annual ACM symposium on Principles of Programming Languages*, 1985.

[20] L. Moreau and C. Queinnec. Partial continuations as the difference of continuations: A duumvirate of control operators. In Manuel Hermenegildo and Jaan Penjam, editors, *PLILP*, pages 182–197. Springer-Verlag, LNCS 844, September 1994.

[21] C. R. Murthy. *Extracting Constructive Content from Classical proofs*. PhD thesis, Cornell University, Department of Computer Science, 1990.

[22] C. R. Murthy. Classical proofs as programs: How, when, and why. Technical Report 91-1215, Cornell University, Department of Computer Science, 1991.

[23] H. Nakano. A constructive logic behind the catch and throw mechanism. *Annals of Pure and Applied Logic*, 69(3):269–301, 1994.

[24] H. Nakano. The non-deterministic catch and throw mechanism and its subject reduction property. In *Logic, Language and Computation*, volume 592 of *LNCS*, pages 61–72. Springer-Verlag, 1994.

[25] H. Nakano. *The Logical Structures of the Catch and Throw Mechanism*. PhD thesis, The University of Tokyo, 1995.

[26] M. Parigot. Free deduction: an analysis of computation in classical logic. In *Proc. Logic Prog. and Autom. Reasoning*, volume 592 of *LNCS*, pages 361–380, 1991.

[27] N. Ramsey. Concurrent programming in ML. Technical Report CS-TR-262-90, Department of Computer Science, Princeton University, Princeton, NJ, 1990.

[28] C. Rauszer. Semi-boolean algebras and their applications to intuitionistic logic with dual operations. In *Fundamenta Mathematicae*, volume 83, pages 219–249, 1974.

[29] C. Rauszer. An algebraic and Kripke-style approach to a certain extension of intuitionistic logic. In *Dissertationes Mathematicae*, volume 167. Institut Mathématique de l'Académie Polonaise des Sciences, 1980.

[30] N. J. Rehof and M. H. Sørensen. The λ_{Δ}-calculus. In *Theoritical Aspects of Computer Software*, volume 542 of *LNCS*, pages 516–542. Springer-Verlag, 1994.

[31] J. H. Reppy. Synchronous operations as first-class values. In David S. Wise, editor, *Proceedings of the SIGPLAN '88 Conference on Programming Lanugage Design and Implementation (SIGPLAN '88)*, pages 250–259, Atlanta, GE, USA, June 1988. ACM Press.

[32] J. H. Reppy. First-class synchronous operations in standard ML. Technical Report TR89-1068, Cornell University, Computer Science Department, December 1989.

[33] J. H. Reppy. Asynchronous signals is standard ML. Technical Report TR90-1144, Cornell University, Computer Science Department, August 1990.

[34] J. H. Reppy. First-class synchronous operations. *Lecture Notes in Computer Science*, 907:235–252, 1995.

[35] M. Sato. Intuitionistic and classical natural deduction systems with the Catch and the Throw rules. *Theoretical Computer Science*, 175(1):75–92, 1997.