Interprétation calculatoire de la logique classique : $\lambda\mu$-calcul et machine de Krivine

Olivier LAURENT
Preuves, Programmes et Systèmes
`Olivier.Laurent@pps.jussieu.fr`

6 septembre 2002

Résumé

À l'aide de la machine de Krivine (une machine abstraite avec pile et environnement pour le λ-calcul), on montrera comment il est possible de compléter le λ-calcul avec des primitives de contrôle (exceptions, `call/cc`, ...), ce qui mène très directement au $\lambda\mu$-calcul de Parigot.

On introduira ensuite les règles de typage du $\lambda\mu$-calcul, ce qui permet d'étendre la correspondance de Curry-Howard à un cadre classique et d'analyser le non déterminisme de la logique classique de Gentzen (LK). On obtient alors des systèmes logiques prouvant les mêmes formules que LK mais dont l'élimination des coupures a une interprétation calculatoire précise.

Mots clefs : λ-calcul, $\lambda\mu$-calcul, machine de Krivine (KAM), déduction naturelle, logique classique, instructions de contrôle, isomorphisme de Curry-Howard.

1 Le λ-calcul

On va se contenter ici de rappeler quelques éléments fondamentaux de λ-calcul. Cette présentation reste volontairement incomplète. Pour une introduction précise et complète au λ-calcul, on pourra se référer au livre de J.-L. Krivine [5].

1.1 Le langage

Étant donné un ensemble dénombrable $\lambda \mathit{Var}$ de variables, les termes du λ-calcul, ou λ-*termes*, sont définis par la grammaire suivante :

$$t \quad ::= \quad x \quad | \quad \lambda x.t \quad | \quad (t)t$$

on notera parfois $(t)t_1 \dots t_n$ comme abréviation pour $(((t)t_1) \dots)t_n$.

Le constructeur λ est un lieur. Les variables libres d'un terme sont donc définies de la manière suivante :

- l'unique variable libre de x est x ;
- les variables libres de $\lambda x.t$ sont celles de t sauf x ;
- les variables libres de $(t)u$ sont celles de t et de u.

On notera $x \in t$ si x est libre dans t. On considère les termes modulo α-renommage des variables liées.

La règle de calcul est la β-réduction :

$$(\lambda x.t)u \quad \to_\beta \quad t[{}^u/_x]$$

qui peut être appliquée n'importe où dans un λ-terme.

Lemme 1 (Réduction et variables libres)
Si $t \to_\beta t'$ alors les variables libres de t' sont incluses dans celles de t.

On peut restreindre les conditions d'application de la β-réduction. On s'intéresse ici à la *réduction de tête faible*. Elle consiste à n'appliquer la β-réduction que si le rédex n'est que sous des applications : $((\lambda x.t)u)u_1 \dots u_n$.

1.2 Les types simples

Les types simples du λ-calcul sont donnés de la manière suivante :

$$A \quad ::= \quad X \quad | \quad A \to A$$

on note $A \to B \to C$ pour le type $A \to (B \to C)$.

En particulier, tout type simple s'écrit d'une unique manière sous la forme $A = A_1 \to \dots \to A_n \to X$.

Un contexte Γ est un ensemble fini de paires (x, A), notées $x : A$, tel que chaque variable n'apparaisse qu'une fois. Un jugement de typage pour le système des types simples est de la forme $\Gamma \vdash t : A$.

Les règles de typage pour le λ-calcul simplement typé sont :

$$\frac{}{x : A \vdash x : A}\,var \qquad \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \to B}\,lam \qquad \frac{\Gamma \vdash t : A \to B \qquad \Delta \vdash u : A}{\Gamma, \Delta \vdash (t)u : B}\,app$$

où Γ, Δ est le contexte obtenu en identifiant les définitions identiques dans Γ et Δ, si $x : A$ apparaît dans Γ et $x : B$ apparaît dans Δ avec $A \neq B$ alors Γ, Δ n'est pas défini.

Il s'agit d'une version *multiplicative* des règles de typage, il en existe de nombreuses autres.

Lemme 2 (Contexte et variables libres)
Si $\Gamma \vdash t : A$ est dérivable, alors Γ contient exactement les variables libres de t.

Proposition 1 (Préservation du type)
Si $\Gamma \vdash t : A$ est dérivable et si $t \to_\beta t'$ alors $\Gamma' \vdash t' : A$ où Γ' est le sous-ensemble de Γ ne contenant que les associations de type pour les variables libres de t'.

2 La machine de Krivine

Afin de donner une alternative à l'évaluation des λ-termes par la β-réduction, on va s'intéresser à la *machine de Krivine* (ou KAM) [6]. Il s'agit d'une machine à pile pour évaluer les λ-termes.

2.1 Définitions et propriétés

On introduit les notions suivantes :
- Un *environnement* e est un fonction partielle à domaine fini de λVar dans l'ensemble des clôtures (qui peut également être représentée par une liste d'associations $x = c$).
- Une *clôture* c est une paire (t, e).
- Une *pile* π est une suite finie (ou pile) de clôtures.
- Un *état* est un triplet $(t\ ,\ e\ ,\ \pi)$ (qui peut être également vu comme une paire $(c\ ,\ \pi)$).

L'intuition est qu'un environnement définit la valeur de certaines variables, une clôture contient un terme et les définitions nécessaires pour ses variables libres sont dans l'environnement et enfin une pile est un contexte d'évaluation.

On utilise les notations suivantes :

- $\emptyset$ est l'environnement vide.
- $e(x) = c$ signifie que la clôture associée à x dans e est c.
- $e + (x = c)$ est l'environnement obtenu en modifiant la valeur associée à x dans e qui devient c.
- Deux environnements e et e' sont dits *compatibles* s'ils ne définissent pas deux valeurs différentes pour la même variable.
- ε est la pile vide.
- $c :: \pi$ est la pile obtenue en ajoutant c en tête (*i.e.* en empilant c sur π).

Les transitions de la KAM décrivent comment passer d'un état à un autre. Le terme t de l'état est considéré comme la suite des *instructions* de la machine.

$$\frac{(t)u \quad\quad e \quad\quad \pi}{t \quad\quad e \quad\quad (u,e) :: \pi} \; \textit{push}$$

$$\frac{\lambda x.t \quad\quad e \quad\quad c :: \pi}{t \quad\quad e + (x = c) \quad\quad \pi} \; \textit{pop}$$

$$\frac{x \quad\quad e \quad\quad \pi}{t \quad\quad e' \quad\quad \pi} \; \textit{deref} \qquad \text{où } e(x) = (t, e')$$

Si aucune transition ne peut être appliquée, la machine s'arrête, et l'état obtenu est le *résultat* du calcul. Deux cas sont possibles : soit le terme est une abstraction et la pile est vide, soit le terme est une variable et l'environnement n'est pas défini pour cette variable.

Si (t, e) est une clôture, on note $t\{e\}$ le *terme* obtenu en substituant dans t chaque variable définie dans e par sa valeur (et en recommençant récursivement puisqu'une telle valeur est à nouveau une clôture). Cette notion est bien définie puisque les environnements sont toujours de profondeur finie et la substitution récursive s'arrête donc. Si π est une pile $(u_1, e_1) :: \ldots :: (u_n, e_n)$, on note $\widetilde{\pi}$ la suite $u_1\{e_1\} \ldots u_n\{e_n\}$.

Propriétés intuitives de la KAM : les propriétés suivantes ne sont pas formelles mais donnent une bonne intuition de ce qui se passe dans l'évaluation de la KAM. Des énoncés plus précis seront donnés ensuite.

- Dans une clôture (t, e), changer la valeur de e pour une variable non libre dans t ne modifie pas le calcul.
- Toute clôture (t, e) peut être remplacée par $(t\{e\}, \emptyset)$ sans que le calcul soit modifié.
- Le calcul de la KAM correspond à la réduction de tête faible et si ce calcul s'arrête, le résultat est la forme normale de tête faible du terme initial.

Exemple 1

$$\begin{array}{ccc|l} (\lambda x.t)u & e & \pi & \\ \hline \lambda x.t & e & (u,e) :: \pi & \textit{push} \\ \hline t & e + (x = (u,e)) & \pi & \textit{pop} \end{array}$$

Lemme 3 (Variables non libres)
Soit (t_0, e_0) une clôture apparaissant dans l'état $(t\ ,\ e\ ,\ \pi)$, si x n'est pas libre dans t_0 alors remplacer

(t_0, e_0) par $(t_0, e_0 + (x = c))$ dans $(t\ ,\ e\ ,\ \pi)$ ne modifie pas les termes obtenus dans le calcul de la KAM.

Démonstration : On montre, pour chaque transition de la machine, que la modification de (t_0, e_0) en $(t_0, e_0 + (x = c))$ donne le même terme et modifie seulement certaines clôtures (t', e') en enrichissant leur environnement e' par des déclarations portant sur des variables non libres dans le terme t'.

(*push*) La transition ne dépend pas de e, les clôtures dans e et π restent inchangées, et si la clôture modifiée est $((t)u, e)$, on ajoute une déclaration dans (u, e) pour une variable qui n'est pas libre dans u.

(*pop*) Modulo α-conversion, on peut supposer que x n'est pas déclarée dans e. Si la clôture modifiée est dans e ou π, elle reste la même, si c'est c la transition n'est pas modifiée non plus et si c'est $(\lambda x.t, e)$ la transition fonctionne de la même manière.

(*deref*) La transition ne dépend que de la valeur de x dans e (et x est libre dans le terme x), rien de particulier concernant les autres clôtures. □

Exemple 2
Une autre règle de réduction que l'on peut ajouter au λ-calcul est la η-réduction :

$$\lambda x.(t)x \quad \to_\eta \quad t \qquad\qquad \text{si } x \notin t$$

cependant elle n'est pas tout à fait réalisée par la KAM : si la pile est vide la machine s'arrête.

$$\lambda x.(t)x \qquad e \qquad \varepsilon$$

sinon le calcul se ramène à celui de t :

$\lambda x.(t)x$	e	$c :: \pi$	
$(t)x$	$e + (x = c)$	π	*pop*
t	$e + (x = c)$	$(x, e + (x = c)) :: \pi$	*push*

Le calcul se poursuit comme si l'on était parti de l'état $(t\ ,\ e\ ,\ c :: \pi)$ (voir lemme 3).

Exemple 3
Si on note $\delta = \lambda x.(x)x$:

$(\delta)\delta$	$\emptyset$	ε	
δ	$\emptyset$	$(\delta, \emptyset)$	*push*
$(x)x$	$x = (\delta, \emptyset)$	ε	*pop*
x	$x = (\delta, \emptyset)$	$(x, x = (\delta, \emptyset))$	*push*
δ	$\emptyset$	$(x, x = (\delta, \emptyset))$	*deref*
	$\dots$		*pop*

le calcul ne termine pas.

Proposition 2 (Simulation)
Si le calcul de la KAM mène de l'état $(t\ ,\ e\ ,\ \pi)$ à l'état $(t'\ ,\ e'\ ,\ \pi')$ alors $(t\{e\})\widetilde{\pi}$ se réduit par réduction de tête faible en $(t'\{e'\})\widetilde{\pi'}$.

Démonstration : On montre que la proposition est valide pour chaque transition de la KAM :

(*push*) On a $((t)u\{e\})\widetilde{\pi} = ((t\{e\})u\{e\})\widetilde{\pi} = (t\{e\})\widetilde{(u, e) :: \pi}$.

(pop) Le terme $(\lambda x.t\{e\})\widetilde{(u,e') :: \pi} = (\lambda x.t\{e\})u\{e'\}\widetilde{\pi}$ se réduit (en réduction de tête faible) en $t\{e\}[^{u\{e'\}}/_x]\widetilde{\pi} = t\{e + x = (u,e')\}\widetilde{\pi}$.

($deref$) Tout simplement $(x\{e\})\widetilde{\pi} = (t\{e'\})\widetilde{\pi}$ si $e(x) = (t,e')$. □

Corollaire 2.1 (Forme normale de tête faible)
Si le calcul de la KAM en partant de l'état $(t\ ,\ \emptyset\ ,\ \varepsilon)$ *s'arrête dans l'état* $(t'\ ,\ e'\ ,\ \pi')$ *alors* $(t'\{e'\})\widetilde{\pi'}$ *est la forme normale de tête faible de* t.

2.2 Typage

Les types de piles sont donnés par :

$$P \quad ::= \quad \bar{X} \quad | \quad A \wedge P$$

où A est un type simple.

On ajoute aux règles de typage du λ-calcul, les règles de typage et de bonne formation pour les environnements, les clôtures, les piles et les états :

$$\frac{}{\vdash_e \emptyset} \qquad \frac{\Gamma\,;\,\Xi \vdash_e e \qquad \Delta \vdash_c c : A}{\Gamma, \Delta\,;\,\Xi, x : A \vdash_e e + (x = c)} \qquad \frac{\Gamma \vdash t : A \qquad \Delta\,;\,\Xi \vdash_e e}{\Gamma \setminus \Xi, \Delta \vdash_c (t,e) : A}$$

$$\frac{}{\vdash_\pi \varepsilon : \bar{X}} \qquad \frac{\Gamma \vdash_c c : A \qquad \Delta \vdash_\pi \pi : P}{\Gamma, \Delta \vdash_\pi (c :: \pi) : A \wedge P}$$

$$\frac{\Gamma \vdash_c (t,e) : A_1 \to \cdots A_n \to X \qquad \Delta \vdash_\pi \pi : A_1 \wedge \cdots \wedge A_n \wedge \bar{X}}{\Gamma, \Delta \vdash_E (t\ ,\ e\ ,\ \pi)}$$

Remarque : On peut également écrire cette dernière règle :

$$\frac{\Gamma \vdash_c (t,e) : A \qquad \Delta \vdash_\pi \pi : A^\perp}{\Gamma, \Delta \vdash_E (t\ ,\ e\ ,\ \pi)}$$

avec : $X^\perp = \bar{X}$ et $(A \to B)^\perp = A \wedge B^\perp$.

$\Gamma \setminus \Xi$ est le contexte obtenu par différence ensembliste entre Γ et Ξ.

La signification des différents jugements utilisés dans les règles ci-dessus est donnée par les deux lemmes suivants.

On dit qu'une variable est libre dans une clôture (t,e) si elle est libre dans t et non définie dans e ou si elle est libre dans l'environnement e. Une variable est libre dans un environnement e si elle est libre dans l'une de ses clôtures.

Lemme 4 (Contexte et variables libres de clôture et d'environnement)
Si $\Gamma \vdash_c c : A$ *est dérivable, alors* Γ *contient exactement les variables libres de* c. *Si* $\Gamma\,;\,\Xi \vdash_e e$ *est dérivable, alors* Γ *contient exactement les variables libres dans* e *et* Ξ *contient exactement les variables définies dans* e.

Une variable est libre dans une pile si elle est libre dans une de ses clôtures.

Lemme 5 (Contexte et variables libres de pile)
Si $\Gamma \vdash_\pi \pi : P$ *est dérivable, alors* Γ *contient exactement les variables libres de* π.

Une variable est libre dans un état si elle est libre dans la clôture ou dans la pile.

Proposition 3 (Préservation du type)
Le typage est préservé par les transitions de la KAM, i.e. *si* $\Gamma \vdash_E (t\ ,\ e\ ,\ \pi)$ *et s'il existe une transition :*

$$\frac{t \qquad e \qquad \pi}{t' \qquad e' \qquad \pi'}$$

alors $\Gamma' \vdash_E (t'\ ,\ e'\ ,\ \pi')$ *où* Γ' *est le sous-ensemble de* Γ *ne contenant que les associations de type pour les variables libres de* $(t'\ ,\ e'\ ,\ \pi')$.

Démonstration : Les seules règles qui ont pu permettre de montrer $\Gamma \vdash_E (t\ ,\ e\ ,\ \pi)$ impliquent que $\Gamma = \Gamma_1 \setminus \Xi, \Gamma_2, \Gamma_3$ avec $\Gamma_1 \vdash t : A$ et $\Gamma_2\,;\,\Xi \vdash_e e$, et $\Gamma_3 \vdash_\pi \pi : A^\perp$. On considère chaque transition :

(*push*) On a $t = (u)v$ donc $\Gamma_1 = \Gamma_1', \Gamma_1''$ avec $\Gamma_1' \vdash u : B \to A$ et $\Gamma_1'' \vdash v : B$, d'où :

$$\dfrac{\dfrac{\Gamma_1' \vdash u : B \to A \qquad \Gamma_2\,;\,\Xi \vdash_e e}{\Gamma_1' \setminus \Xi, \Gamma_2 \vdash_c (u,e) : B \to A} \qquad \dfrac{\dfrac{\Gamma_1'' \vdash v : B \qquad \Gamma_2\,;\,\Xi \vdash_e e}{\Gamma_1'' \setminus \Xi, \Gamma_2 \vdash_c (v,e) : B} \qquad \Gamma_3 \vdash_\pi \pi : A^\perp}{\Gamma_1'' \setminus \Xi, \Gamma_2, \Gamma_3 \vdash_\pi ((v,e) :: \pi) : B \wedge A^\perp}}{\Gamma_1' \setminus \Xi, \Gamma_1'' \setminus \Xi, \Gamma_2, \Gamma_3 \vdash_E (t'\ ,\ e'\ ,\ \pi')}$$

(*pop*) On a $t = \lambda x.t'$ donc $A = B \to C$ et $\Gamma_1, x : B \vdash t' : C$, de plus $\pi = c :: \pi'$ donc $\Gamma_3 = \Gamma_3', \Gamma_3''$ avec $\Gamma_3' \vdash_c c : B$ et $\Gamma_3'' \vdash_\pi \pi' : C^\perp$, d'où :

$$\dfrac{\dfrac{\Gamma_1, x : B \vdash t' : C \qquad \dfrac{\Gamma_2\,;\,\Xi \vdash_e e \qquad \Gamma_3' \vdash_c c : B}{\Gamma_2, \Gamma_3'\,;\,\Xi, x : B \vdash_e e + (x = c)}}{\Gamma_1 \setminus \Xi, \Gamma_2, \Gamma_3' \vdash_c (t', e + (x = c)) : C} \qquad \Gamma_3'' \vdash_\pi \pi' : C^\perp}{\Gamma_1 \setminus \Xi, \Gamma_2, \Gamma_3', \Gamma_3'' \vdash_E (t'\ ,\ e'\ ,\ \pi')}$$

(*deref*) On a $t = x$ donc $\Gamma_1 = x : A$ et $e = e_0 + (x = (t', e'))$. Par typage de e, on déduit que $\Gamma_2 = \Gamma_2', \Gamma_2'', x : A$ avec $\Gamma_2'' \vdash_c (t', e') : A$. D'où, finalement :

$$\frac{\Gamma_2'' \vdash_c (t', e') : A \qquad \Gamma_3 \vdash_\pi \pi : A^\perp}{\Gamma_2'', \Gamma_3 \vdash_E (t'\ ,\ e'\ ,\ \pi')}$$

□

Corollaire 3.1 (Terminaison)
Si $\Gamma \vdash_E (t\ ,\ e\ ,\ \pi)$*, le calcul de la KAM en partant de* $(t\ ,\ e\ ,\ \pi)$ *s'arrête dans un état* $(x\ ,\ e'\ ,\ \pi')$ *et* x *est la variable de tête de la forme normale de tête de* $(t\{e\})\pi$.

3 KAM et instructions de contrôle

3.1 Extension de la machine

Si l'on considère les états de la KAM comme une paire (c, π), les instructions de la machine permettent de gérer des clôtures : mettre une clôture sur la pile (*push*), nommer une clôture dans l'environnement (*pop*) et lire une clôture dans l'environnement (*deref*).

Remarque : Les piles permettent de représenter tous les contextes dans le cas du λ-calcul, comme tend à l'indiquer le théorème de Böhm :

> Soient t et t' deux λ-termes clos et x et y deux variables, on a $t \not\simeq_{\beta\eta} t'$ *ssi* il existe un contexte $C[.]$ tel que $C[t] \to_\beta x$ et $C[t'] \to_\beta y$ *ssi* il existe des termes u_1, …, u_n tels que $(t)u_1 \ldots u_n \to_\beta x$ et $(t')u_1 \ldots u_n \to_\beta y$.

On va étendre la KAM avec des instructions permettant de manipuler les piles comme on a manipulé les clôtures jusqu'ici. Pour cela on étend le langage des instructions, c'est-à-dire le langage des termes :

$$t \quad ::= \quad x \quad | \quad \lambda x.t \quad | \quad (t)t \quad | \quad \mu\alpha.t \quad | \quad [\alpha]t$$

On introduit un nouvel ensemble de variables (les μ-variables) notées α, β, ... Le constructeur μ est un lieur pour ces μ-variables. Ainsi les μ-variables libres d'un terme sont obtenues de la manière suivante :

- il n'y a pas de μ-variable libre dans x ;
- les μ-variables libres de $\lambda x.t$ sont celles de t ;
- les μ-variables libres de $(t)u$ sont celles de t et de u ;
- les μ-variables libres de $\mu\alpha.t$ sont celles de t sauf α ;
- les μ-variables libres de $[\alpha]t$ sont celles de t plus α.

On considère les termes modulo α-renommage également sur les μ-variables liées.

Un environnement contiendra désormais deux types d'informations, des clôtures associées aux λ-variables et des piles associées aux μ-variables.

Ceci nous donne deux nouvelles transitions dans la KAM :

$$\frac{\mu\alpha.t \qquad e \qquad \pi}{t \qquad e + (\alpha = \pi) \qquad \varepsilon} \; \textit{save}$$

$$\frac{[\alpha]t \qquad e \qquad \varepsilon}{t \qquad e \qquad \pi} \; \textit{restore} \qquad \text{où } e(\alpha) = \pi$$

La machine peut désormais s'arrêter avant une transition (*restore*) parce que la pile n'est pas vide ou parce que α n'est pas défini dans l'environnement.

Lemme 6 (Variables non libres)
Soit (t_0, e_0) une clôture apparaissant dans l'état $(t\ ,\ e\ ,\ \pi)$, si α n'est pas libre dans t_0 alors remplacer (t_0, e_0) par $(t_0, e_0 + (\alpha = \pi_0))$ dans $(t\ ,\ e\ ,\ \pi)$ ne modifie pas les termes obtenus dans le calcul de la KAM.

Démonstration : Les deux seuls cas intéressants sont ceux des nouvelles transitions :

(*save*) Modulo α-conversion, on peut supposer que α n'est pas déclarée dans e, et la transition ne dépend pas des autres déclarations de variables.

(*restore*) La transition ne dépend que de la valeur de α dans e et α est libre dans $[\alpha]t$. □

Exemple 4

$$\begin{array}{ccc|l} (\mu\alpha.t)u & e & \varepsilon & \\ \hline \mu\alpha.t & e & (u,e) & \textit{push} \\ \hline t & e + (\alpha = (u,e)) & \varepsilon & \textit{save} \end{array}$$

De manière plus générale, si on note $\vec{u}$ la suite $u_1 \dots u_n$ et $\pi_{\vec{u}}$ la pile $(u_1, e) :: \dots :: (u_n, e)$.

$$\begin{array}{ccc|l} (\mu\alpha.t)\vec{u} & e & \varepsilon & \\ \hline & \vdots & & \textit{push} \\ \hline \mu\alpha.t & e & \pi_{\vec{u}} & \textit{push} \\ \hline t & e + (\alpha = \pi_{\vec{u}}) & \varepsilon & \textit{save} \end{array}$$

Exemple 5

$$\begin{array}{ccc} (\mu\alpha.t)u & e & \pi \\ \hline \mu\alpha.t & e & (u,e) :: \pi \\ \hline t & e + (\alpha = (u,e) :: \pi) & \varepsilon \end{array} \qquad \begin{array}{l} \textit{push} \\ \textit{save} \end{array}$$

et

$$\begin{array}{ccc} \mu\alpha.t[{}^{[\alpha](v)u}/_{[\alpha]v}] & e & \pi \\ \hline t[{}^{[\alpha](v)u}/_{[\alpha]v}] & e + (\alpha = \pi) & \varepsilon \end{array} \qquad \textit{save}$$

La notation $t[{}^{[\alpha](v)u}/_{[\alpha]v}]$ signifie que tous les sous-termes de t commençant par $[\alpha]$, donc de la forme $[\alpha]v$ pour un certain v, sont remplacés par $[\alpha](v)u$. Par exemple :

$$\lambda f.\mu\alpha[\alpha](f)\lambda x.\mu\delta[\alpha]x[{}^{[\alpha](v)u}/_{[\alpha]v}] = \lambda f.\mu\alpha[\alpha]((f)\lambda x.\mu\delta[\alpha](x)u)u$$

Exemple 6
Faire successivement les deux opérations $\mu\alpha$ et $[\alpha]$ a pour effet de continuer le calcul avec la même pile π et le même environnement e, excepté que désormais la variable α vaut π dans l'environnement :

$$\begin{array}{ccc} \mu\alpha[\alpha]t & e & \pi \\ \hline [\alpha]t & e + (\alpha = \pi) & \varepsilon \\ \hline t & e + (\alpha = \pi) & \pi \end{array} \qquad \begin{array}{l} \textit{save} \\ \textit{restore} \end{array}$$

3.2 Instructions de contrôle

On cherche à se convaincre que les nouvelles instructions que l'on a ajoutées à la KAM permettent de réaliser des instructions de contrôle. Pour cela on se donne une extension du λ-calcul avec des exceptions :

$$t \quad ::= \quad x \quad | \quad \lambda x.t \quad | \quad (t)t \quad | \quad \texttt{raise}\ k \quad | \quad \texttt{try}\ t\ \texttt{with}\ k \mapsto u$$

L'intuition est la suivante : on calcule le terme en réduction de tête faible, si on rencontre un `raise` k en dehors de tout `try` on s'arrête et si, par contre, on rentre dans un `try`, on calcule t normalement sauf si on rencontre un `raise` k auquel cas on interrompt le calcul pour démarrer celui de u avec le même contexte qu'au moment de l'entrée dans le `try`.

On ne rentrera volontairement pas dans une description formelle et précise de la sémantique de ce langage avec exceptions, pour plus de détails on pourra se référer au cas de ML (par exemple [12]).

On code ce langage dans notre langage d'instructions de la KAM de la manière suivante :

$$\begin{array}{ccll} \overline{x}^{\varphi} & = & x & \\ \overline{\lambda x.t}^{\varphi} & = & \lambda x.\overline{t}^{\varphi} & \\ \overline{(t)u}^{\varphi} & = & (\overline{t}^{\varphi})\overline{u}^{\varphi} & \\ \overline{\texttt{raise}\ k}^{\varphi} & = & (k)u & \text{où } \varphi(k) = u \\ \overline{\texttt{try}\ t\ \texttt{with}\ k \mapsto u}^{\varphi} & = & (\lambda f.\mu\alpha[\alpha](f)\lambda x.\mu\delta[\alpha]x)\lambda k.\overline{t}^{\varphi+(k=u)} & \end{array}$$

où φ est un environnement qui associe des termes aux variables d'exceptions.

Remarque : On constate que l'on a la possibilité de changer le u à chaque occurrence de `raise` k, ceci est codable dans le langage source si les exceptions transportent un paramètre comme c'est le cas en ML.

On peut voir ce que donne le calcul de la KAM pour la traduction d'un terme de la forme `try` t `with` $k \mapsto u$ (on note $\overline{t}$ pour $\overline{t}^{k=u}$) :

Terme	Environnement	Pile	Transition
$(\lambda f.\mu\alpha[\alpha](f)\lambda x.\mu\delta[\alpha]x)\lambda k.\overline{t}$	e	π	*push*
$\lambda f.\mu\alpha[\alpha](f)\lambda x.\mu\delta[\alpha]x$	e	$(\lambda k.\overline{t}, e) :: \pi$	*pop*
$\mu\alpha[\alpha](f)\lambda x.\mu\delta[\alpha]x$	$e + (f = (\lambda k.\overline{t}, e))$	π	*save*
$[\alpha](f)\lambda x.\mu\delta[\alpha]x$	$e + (f = (\lambda k.\overline{t}, e)) + (\alpha = \pi)$	ε	*restore*
$(f)\lambda x.\mu\delta[\alpha]x$	$e + (f = (\lambda k.\overline{t}, e)) + (\alpha = \pi)$	π	*push*
f	$e + (f = (\lambda k.\overline{t}, e)) + (\alpha = \pi)$	$(\lambda x.\mu\delta[\alpha]x, e') :: \pi$	*deref*
$\lambda k.\overline{t}$	e	$(\lambda x.\mu\delta[\alpha]x, e') :: \pi$	*pop*
$\overline{t}$	$e + (k = (\lambda x.\mu\delta[\alpha]x, e'))$	π	

où $e' = e + (f = (\lambda k.\overline{t}, e)) + (\alpha = \pi)$. On peut remplacer $(\lambda x.\mu\delta[\alpha]x, e')$ par $(\lambda x.\mu\delta[\alpha]x, \alpha = \pi)$ en vertu des lemmes 3 et 6.

Si t ne contient pas de `raise` k, le calcul se passe comme pour t puisque $k \notin \overline{t}$.

Sinon, on considère le cas où k est la seule variable libre de t ce qui a pour conséquence que l'on peut remplacer e par $\emptyset$ dans le calcul précédent. Si la suite du calcul aboutit sur un `raise` k, la machine effectue les étapes suivantes (où e_0 contient $k = (\lambda x.\mu\delta[\alpha]x, \alpha = \pi)$) :

Terme	Environnement	Pile	Transition
$(k)u$	e_0	π_0	*push*
k	e_0	$(u, e_0) :: \pi_0$	*deref*
$\lambda x.\mu\delta[\alpha]x$	$\alpha = \pi$	$(u, e_0) :: \pi_0$	*pop*
$\mu\delta[\alpha]x$	$(\alpha = \pi) + (x = (u, e_0))$	π_0	*save*
$[\alpha]x$	$(\alpha = \pi) + (x = (u, e_0)) + (\delta = \pi_0)$	ε	*restore*
x	$(\alpha = \pi) + (x = (u, e_0)) + (\delta = \pi_0)$	π	*deref*
u	e_0	π	

on aboutit bien au calcul de u avec comme pile, celle présente lors de l'entrée dans le `try`.

4 $\lambda\mu$-calcul

On va désormais se restreindre au cas où $\mu\alpha$ et $[\beta]$ sont toujours collés ce qui limite les cas de blocage de la machine, puisque la pile est ainsi nécessairement vide lors d'une transition (*restore*). De plus, dans tous les exemples que l'on a vus, cette propriété est vérifiée.

Le langage que l'on obtient est alors précisément le $\lambda\mu$-calcul [9] de M. Parigot, une extension du λ-calcul permettant de représenter des primitives de contrôle.

4.1 Le langage

On se donne deux ensembles dénombrables disjoints λVar et μVar de variables : les λ-variables x, y, ... et les μ-variables α, β, ...

$$t \quad ::= \quad x \quad | \quad \lambda x.t \quad | \quad (t)t \quad | \quad \mu\alpha[\beta]t$$

Rappelons que, tout comme λ est un lieur pour les λ-variables, μ est un lieur pour les μ-variables. Et si $\alpha \neq \beta$, la variable β est libre dans $\mu\alpha[\beta]t$.

La principale règle de calcul est la règle de μ-réduction :

$$(\mu\alpha.t)u \quad \to_\mu \quad \mu\alpha.t[{}^{[\alpha](v)u}/_{[\alpha]v}]$$

Pour la notation $t[^{[\alpha](v)u}/_{[\alpha]v}]$, voir l'exemple 5.

On peut également ajouter les deux règles :

$$\begin{array}{lll} [\beta]\mu\alpha.t & \rightarrow_\rho & t[^\beta/_\alpha] \\ \mu\alpha[\alpha]t & \rightarrow_\theta & t \qquad\qquad \text{si } \alpha \notin t \end{array}$$

On notera $u \rightarrow v$ pour $u \rightarrow_\beta v$ ou $u \rightarrow_\mu v$ ou $u \rightarrow_\rho v$.

Exemple 7

$$\begin{array}{ccc|l} [\beta]\mu\alpha.t & e + (\beta = \pi) & \varepsilon & \\ \hline \mu\alpha.t & e + (\beta = \pi) & \pi & \textit{restore} \\ \hline t & e + (\beta = \pi) + (\alpha = \pi) & \varepsilon & \textit{save} \end{array}$$

ce qui valide la règle ρ si l'environnement contient des déclarations pour toutes les variables libres du terme au départ.

Exemple 8
Pour valider la règle θ, on considère le cas particulier de l'exemple 6 où $\alpha \notin t$:

$$\begin{array}{ccc|l} \mu\alpha[\alpha]t & e & \pi & \\ \hline [\alpha]t & e + (\alpha = \pi) & \varepsilon & \textit{save} \\ \hline t & e + (\alpha = \pi) & \pi & \textit{restore} \end{array}$$

dont le calcul continue comme pour $(t\ ,\ e\ ,\ \pi)$, d'après le lemme 6.

Proposition 4 (Confluence)
Si $t \rightarrow^* u$ et $t \rightarrow^* v$ alors il existe w tel que $u \rightarrow^* w$ et $v \rightarrow^* w$.

DÉMONSTRATION : Voir [9]. □

La réduction de tête faible pour le $\lambda\mu$-calcul accepte que l'on réduise un rédex sous des applications et sous des constructions $\mu\alpha[\beta]$.

On enrichit la définition de $t\{e\}$ pour le cas où e contient des associations $\alpha = \pi$ en substituant dans t toute occurrence de $[\alpha]u$ par $(u)\widetilde{\pi}$.

Proposition 5 (Simulation par la KAM)
Si le calcul de la KAM mène de l'état $(t\ ,\ e\ ,\ \pi)$ avec (t, e) close à l'état $(t'\ ,\ e'\ ,\ \pi')$ alors $(t\{e\})\widetilde{\pi}$ se réduit par réduction de tête faible en un terme t_0 tel que $(t'\{e'\})\widetilde{\pi'}$ est obtenu en effaçant dans t_0 des $\mu\alpha$ en tête et les $[\alpha]$ liés par ces $\mu\alpha$.

DÉMONSTRATION : On montre que la proposition est valide pour les nouvelles transitions de la KAM :

(*save*) $((\mu\alpha.t)\{e\})\widetilde{\pi} = (\mu\alpha.(t\{e\}))\widetilde{\pi}$ se réduit en plusieurs étapes en $\mu\alpha.(t\{e + \alpha = \pi\})$ qui donne $t\{e + \alpha = \pi\}$ en enlevant le $\mu\alpha$ de tête et les $[\alpha]$ correspondants.

(*restore*) Par définition, $([\alpha]t)\{e\} = (t\{e\})\widetilde{\pi}$. □

4.2 Les types simples

On cherche à construire des règles de typage pour le $\lambda\mu$-calcul qui soient cohérentes avec l'évaluation de la KAM. On va se baser sur le cas suivant :

$$\begin{array}{ccc|l} \mu\beta[\alpha]t & \alpha = \pi' & \pi & \\ \hline [\alpha]t & (\alpha = \pi') + (\beta = \pi) & \varepsilon & \textit{save} \\ \hline t & (\alpha = \pi') + (\beta = \pi) & \pi' & \textit{restore} \end{array}$$

Supposons que $t : A$, d'après le dernier état on a $\pi' : A^\perp$. Puisque l'environnement contient la déclaration $\alpha = \pi'$, le type de α doit être en correspondance avec celui de π', et puisque l'on veut rester dans les types simples, on va essayer de choisir $\alpha : A$. Soit B le type de β, d'après l'environnement, on a $\pi : B^\perp$ et donc d'après le premier état, $\mu\beta[\alpha]t : B$. Ceci nous donnerait intuitivement une règle de typage de la forme :

$$\frac{\beta : B \text{ et } t : A}{\alpha : A \text{ et } \mu\beta[\alpha]t : B}\, mu$$

Si on compare avec la règle *lam* écrite de la même manière :

$$\frac{x : A \text{ et } t : B}{\lambda x.t : A \to B}\, lam$$

on constate une différence importante, A apparaît dans $A \to B$ comme étant en occurrence négative, ce qui justifie que la déclaration $x : A$ apparaisse à gauche dans le jugement de typage $x : A \vdash t : B$ de la règle formelle. Afin de formaliser la règle *mu*, on voit que A et B doivent apparaître à droite puisqu'ils apparaissent en position positive dans le type de t. C'est pourquoi on va considérer des jugements de typage avec un emplacement à gauche pour les λ-variables et un emplacement à droite pour les μ-variables :

$$x_1 : A_1, \ldots, x_n : A_n \vdash t : A \mid \alpha_1 : B_1, \ldots, \alpha_m : B_m \qquad \text{noté aussi} \qquad \Gamma \vdash t : A \mid \Delta$$

Ce qui nous mène à la règle *mu* formelle :

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

Avec deux cas particuliers : β peut ne pas être présent dans le contexte de la prémisse (comme c'est le cas pour x dans *lam*) et α peut déjà être présent dans le contexte si α est libre dans t, ce qui donne :

$$\frac{\Gamma \vdash t : A \mid \Delta}{\Gamma \vdash \mu\beta[\alpha]t : B \mid \alpha : A, \Delta} \qquad \frac{\Gamma \vdash t : A \mid \beta : B, \alpha : A, \Delta}{\Gamma \vdash \mu\beta[\alpha]t : B \mid \alpha : A, \Delta}$$

Ainsi, la règle *mu* respecte le fait que les variables présentes dans le contexte sont exactement les variables libres dans t. Si β était libre dans t alors β ne l'est plus dans $\mu\beta[\alpha]t$ et si $\alpha \neq \beta$ alors α est libre dans $\mu\beta[\alpha]t$.

Proposition 6 (Préservation du type)
Si $\Gamma \vdash t : A \mid \Delta$ est dérivable et si $t \to t'$ alors $\Gamma' \vdash t' : A \mid \Delta'$ où Γ' (resp. Δ') est le sous-ensemble de Γ (resp. Δ) ne contenant que les associations de type pour les λ-variables (resp. μ-variables) libres de t'.

DÉMONSTRATION : Voir [9]. □

Proposition 7 (Normalisation forte)
Si t est un $\lambda\mu$-terme typable, alors t n'admet pas de suite de réductions infinie.

DÉMONSTRATION : Voir [11]. □

Remarque : En ajoutant un type $\perp$ pour les termes, on peut de nouveau séparer $\mu\alpha$ et $[\beta]$. En effet, le typage peut nous garantir que la pile doit être vide au bon moment. On utilise alors les règles suivantes pour le typage de $\mu\alpha$ et $[\beta]$:

$$\frac{\Gamma \vdash t : A \mid \Delta}{\Gamma \vdash [\alpha]t : \perp \mid \alpha : A, \Delta} \qquad \frac{\Gamma \vdash t : \perp \mid \beta : B, \Delta}{\Gamma \vdash \mu\beta.t : B \mid \Delta}$$

Ceci a pour conséquence d'enrichir strictement le langage puisque le terme :

$$\lambda f^{(A\to\bot)\to\bot}.\mu\alpha^A.(f)\lambda x^A.[\alpha]x : ((A\to\bot)\to\bot)\to A$$

est alors correct et typable.

Remarque : On peut étendre le typage de la KAM pour le cas du $\lambda\mu$-calcul en ajoutant des contextes à droite pour les μ-variables et la règle suivante :

$$\frac{\Gamma\,;\,\Xi\vdash_e e \mid \Theta\,;\,\Delta \qquad \Gamma'\vdash_\pi \pi : A^\bot \mid \Delta'}{\Gamma,\Gamma'\,;\,\Xi\vdash_e e+(\alpha=\pi) \mid \alpha : A,\Theta\,;\,\Delta,\Delta'}$$

5 Correspondance de Curry-Howard

On va montrer comment le $\lambda\mu$-calcul et la déduction naturelle classique permettent d'étendre à un cadre classique la correspondance de Curry-Howard intuitionniste entre λ-calcul simplement typé et déduction naturelle pour la logique minimale.

5.1 Déduction naturelle classique

La déduction naturelle permet de prouver les formules valides en logique intuitionniste. Afin d'obtenir un système pour les formules classiques, M. Parigot a introduit la déduction naturelle classique [9] qui étend la déduction naturelle avec plusieurs conclusions.

Les séquents de la déduction naturelle sont étendus avec un contexte à droite et les nouvelles règles sont obtenues en supprimant les termes et les variables dans les règles de typage du $\lambda\mu$-calcul, dans le cadre de la correspondance de Curry-Howard.

Il suffit en fait de considérer séparément les deux cas particuliers de la règle *mu* :

$$\frac{\Gamma\vdash\Delta}{\Gamma\vdash A,\Delta}\,\mathit{aff}_d \qquad \frac{\Gamma\vdash A,A,\Delta}{\Gamma\vdash A,\Delta}\,\mathit{ctr}_d$$

Pour prouver que l'on obtient la prouvabilité classique, on démontre la loi de Peirce $((A\to B)\to A)\to A$.

$$\cfrac{\cfrac{\cfrac{}{(A\to B)\to A\vdash(A\to B)\to A}\,ax \qquad \cfrac{\cfrac{\cfrac{}{A\vdash A}\,ax}{A\vdash B,A}\,\mathit{aff}_d}{\vdash A\to B,A}\,\to_i}{\cfrac{(A\to B)\to A\vdash A,A}{(A\to B)\to A\vdash A}\,\mathit{ctr}_d}\,\to_e}{\vdash((A\to B)\to A)\to A}\,\to_i$$

ce qui correspond au $\lambda\mu$-terme typé : $\lambda f^{(A\to B)\to A}.\mu\alpha^A[\alpha](f)\lambda x^A.\mu\delta^B[\alpha]x : ((A\to B)\to A)\to A$.

Il reste à définir la normalisation des preuves. Un exemple caractéristique des nouveaux cas qui apparaissent est :

$$\frac{\dfrac{\Gamma\vdash A\to B,A\to B,\Delta}{\Gamma\vdash A\to B,\Delta}\,\mathit{ctr}_d \qquad \dfrac{\Gamma'\vdash A,A,\Delta'}{\Gamma'\vdash A,\Delta'}\,\mathit{ctr}_d}{\Gamma,\Gamma'\vdash B,\Delta,\Delta'}\,\to_e$$

qui correspond au $\lambda\mu$-terme $(\mu\alpha.t)\mu\beta.u$, on va suivre la réduction donnée par le $\lambda\mu$-calcul :

$$(\mu\alpha.t)\mu\beta.u \quad \to_\mu \quad \mu\alpha.t[{}^{[\alpha](v)\mu\beta.u}/_{[\alpha]v}]$$

(la fonction duplique d'abord l'argument).

De manière générale, si la preuve n'est pas en forme normale on regarde une règle $\to_e$ qui viole la propriété de la sous-formule. Soit la règle qui précède est une règle $\to_i$ et on applique l'étape usuelle, soit c'est une règle structurelle et dans ce cas on applique la nouvelle étape :

$$\dfrac{\dfrac{\begin{array}{c}\cdots \quad \dfrac{\dfrac{\pi_i}{\Gamma_i, A \vdash B, \Delta_i}}{\Gamma_i \vdash A \to B, \Delta_i}\to_i \quad \cdots \\ \vdots \\ \Gamma \vdash A \to B, \Delta\end{array} \qquad \dfrac{\pi}{\Gamma' \vdash A, \Delta'}}{\Gamma, \Gamma' \vdash B, \Delta, \Delta'}\to_e}{}$$

$$\downarrow$$

$$\begin{array}{c}\cdots \quad \dfrac{\dfrac{\dfrac{\pi_i}{\Gamma_i, A \vdash B, \Delta_i}}{\Gamma_i \vdash A \to B, \Delta_i}\to_i \qquad \dfrac{\pi}{\Gamma' \vdash A, \Delta'}}{\Gamma_i, \Gamma' \vdash B, \Delta_i, \Delta'}\to_e \quad \cdots \\ \vdots \\ \Gamma, \Gamma' \vdash B, \Delta, \Delta'\end{array}$$

la preuve π remonte le long de la preuve de $A \to B$ jusqu'à chaque endroit où une règle $\to_i$ introduisant $A \to B$ a été appliquée.

Remarque : Il est également possible de définir une version en appel par valeur du $\lambda\mu$-calcul [9, 8], qui correspond à effectuer la réduction dans l'autre sens : l'argument duplique d'abord la fonction.

5.2 Calcul des séquents

Dans le cas du calcul des séquents LK de Gentzen, le problème de la réduction est plus délicat dans un cas comme celui-ci :

$$\dfrac{\dfrac{\Gamma \vdash A, A, \Delta}{\Gamma \vdash A, \Delta}\,ctr_d \qquad \dfrac{\Gamma', A, A \vdash \Delta'}{\Gamma', A \vdash \Delta'}\,ctr_g}{\Gamma, \Gamma' \vdash \Delta, \Delta'}\,cut$$

ou, dans un calcul monolatère :

$$\dfrac{\dfrac{\vdash \Gamma, A, A}{\vdash \Gamma, A}\,ctr \qquad \dfrac{\vdash \Gamma', \neg A, \neg A}{\vdash \Gamma', \neg A}\,ctr}{\vdash \Gamma, \Gamma'}\,cut$$

La situation est alors symétrique entre les deux preuves coupées, on n'a plus de distinction fonction/argument ce qui empêche de définir une élimination déterministe des coupures. Afin de briser cette symétrie, J.-Y. Girard a introduit le système LC [4] qui est une variante de LK basée sur une notion de polarités. Chaque formule est soit positive, soit négative et la négation échange la polarité. Ceci permet de briser la symétrie puisque dans une coupure on a exactement une formule coupée positive. On décide alors, de manière uniforme, que la preuve contenant la formule coupée positive commence par dupliquer l'autre.

6 D'autres lectures

Pour plus d'informations sur le $\lambda\mu$-calcul, on peut lire l'article d'origine [9] et deux autres articles de M. Parigot sur l'utilisation calculatoire du $\lambda\mu$-calcul (en particulier à propos des entiers) [10] et sur la normalisation forte [11].

Le $\lambda\mu$-calcul en appel par valeur a été introduit en détails par L. Ong et C. Stewart [8]. Un travail similaire à celui que nous présentons ici a été fait pour l'appel par valeur par G. Bierman [1].

Les articles suivants sont directement ou indirectement liés à la Logique Linéaire. V. Danos et L. Regnier ont montré comment retrouver la KAM à partir de la géométrie de l'interaction [3]. V. Danos, J.-B. Joinet et H. Schellinx ont fait une étude très détaillée de l'élimination des coupures en logique classique de manière à la rendre déterministe [2]. O. Laurent a montré le lien entre LC et le $\lambda\mu$-calcul et a défini une équivalence sur les $\lambda\mu$-termes qui est compatible avec le calcul dans la KAM [7].

Merci à P. Baillot, C. Fouqueré et J. Vauzeilles pour m'avoir encouragé à écrire ces notes et à C. Berline pour m'avoir donné l'occasion de le faire.

Références

[1] Gavin Bierman. A computational interpretation of the $\lambda\mu$-calculus. In L. Brim, J. Gruska, and J. Zlatuska, editors, *Proceedings of Mathematical Foundations of Computer Science*, volume 1450 of *Lecture Notes in Computer Science*, pages 336–345. Springer, August 1998.

[2] Vincent Danos, Jean-Baptiste Joinet, and Harold Schellinx. A new deconstructive logic : linear logic. *Journal of Symbolic Logic*, 62(3) :755–807, September 1997.

[3] Vincent Danos and Laurent Regnier. Reversible, irreversible and optimal λ-machines. *Theoretical Computer Science*, 227 :79–97, September 1999.

[4] Jean-Yves Girard. A new constructive logic : classical logic. *Mathematical Structures in Computer Science*, 1(3) :255–296, 1991.

[5] Jean-Louis Krivine. *Lambda-Calcul : Types et Modèles.* Etudes et Recherches en Informatique. Masson, 1990.

[6] Jean-Louis Krivine. Un interpréteur du lambda-calcul. Available at `ftp ://ftp.logique.jussieu.fr/pub/distrib/krivine/interprt.pdf` , 1992.

[7] Olivier Laurent. Polarized proof-nets and $\lambda\mu$-calculus. *Theoretical Computer Science*, 290(1) :161–188, January 2003.

[8] Luke Ong and Charles Stewart. A Curry-Howard foundation for functional computation with control. In *Proceedings of the 24^{th} Annual ACM Symposium on Principles of Programming Languages*, pages 215–227. Association for Computing Machinery, ACM Press, January 1997.

[9] Michel Parigot. $\lambda\mu$-calculus : an algorithmic interpretation of classical natural deduction. In *Proceedings of Internationnal Conference on Logic Programming and Automated Deduction*, volume 624 of *Lecture Notes in Computer Science*, pages 190–201. Springer, 1992.

[10] Michel Parigot. Classical proofs as programs. In *Proceedings of Kurt Gödel Colloquium*, volume 713 of *Lecture Notes in Computer Science*, pages 263–276. Springer, 1993.

[11] Michel Parigot. Strong normalization for second order classical natural deduction. In *Proceedings of the heigth annual IEEE symposium on Logic In Computer Science*, pages 39–46. IEEE Computer Society Press, 1993.

[12] Didier Rémy. Typage et programmation. Notes de cours du D.E.A. de programmation. Available at `http ://pauillac.inria.fr/~remy/classes/dea/` , January 1998.