Krivine's abstract machine and the $\lambda\mu$-calculus

(an overview)

Olivier LAURENT

Preuves, Programmes et Systèmes

CNRS – Université Paris VII

UMR 7126 – Case 7014

175, rue du Chevaleret – 75013 Paris – FRANCE

Olivier.Laurent@pps.jussieu.fr

September 17, 2003

Abstract

After a presentation of Krivine's abstract machine for the pure and simply typed λ-calculi, we show how an extension of the instructions for the manipulation of stacks leads to Parigot's $\lambda\mu$-calculus. Using a typing system for the machine, we derive the typing rules for the simply typed $\lambda\mu$-calculus.

Keywords : λ-calculus, $\lambda\mu$-calculus, Krivine's abstract machine (KAM), classical logic, control operators.

Introduction

M. Parigot introduced the *$\lambda\mu$-calculus* [14] as an extension of the λ-calculus allowing to extend the Curry-Howard correspondence to classical logic, in the spirit of Griffin's ideas [8]. His work is based on a proof theoretical approach in the study of natural deduction with many conclusions. We propose to show how it is possible to define and describe the $\lambda\mu$-calculus with more operational considerations. We will use *Krivine's abstract machine* (KAM) [10] as computational framework, presented in a first step as an abstract machine for the λ-calculus. The definition of a typing system for the machine allows to control some properties of the machine: stopping states, termination, ... In a second step, we extend the machine with instructions for the manipulation of stacks which appear to correspond precisely to the $\lambda\mu$-calculus and which give enough expressive power to encode control primitives (like call/cc, continuations, ...). The typing system of the machine allows then to derive the typing rules of the simply typed $\lambda\mu$-calculus. So that Parigot's $\lambda\mu$-calculus is entirely rebuilt from Krivine's abstract machine.

We can almost say that none of the ideas introduced in this paper is due to the author. These are common ideas (very well known by M. Parigot and J.-L. Krivine themselves) that have never been written in details, as far as we know. Our goal is to give an alternative presentation of the $\lambda\mu$-calculus with respect to Parigot's one [14], that can be used by people with a basic knowledge in λ-calculus.

1 The λ-calculus

We are just going to recall some elements of λ-calculus in order to introduce some notations and to present the results we will use. For a more precise and complete introduction to the λ-calculus, see [1, 11].

1.1 The language

Let λVar be a denumerable set of variables called the λ-variables and denoted x, y, z, ..., the λ-terms are given by:

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

we will use the notation $(t)t_1 \dots t_n$ for $(((t)t_1)\dots)t_n$.

Remark: These kinds of notations with brackets around the function of an application are particularly natural for Krivine's Abstract Machine since $(t)t_1 \dots t_n$ will be an instruction which first builds the stack $t_1 \dots t_n$ and then executes t.

The λ-construction is a binder for the λ-variable x. The *free* variables of a term (denoted by $x \in t$) are thus defined by:

- x has a unique free variable x;
- the free variables of $\lambda x.t$ are those of t except x;
- the free variables of $(t)u$ are those of t and u.

Terms are used up to α-equivalence for bound variables.

The β-reduction is the only computation rule and can be applied anywhere in a λ-term:

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

where $t[{}^u/_x]$ is the usual capture-avoiding substitution of variables by λ-terms defined by: $x[{}^u/_x] = u$, $y[{}^u/_x] = y$ if $x \neq y$, $((t)t')[{}^u/_x] = (t[{}^u/_x])t'[{}^u/_x]$ and $(\lambda y.t)[{}^u/_x] = \lambda y.(t[{}^u/_x])$ where y is chosen not free in u and x (using α-equivalence).

Lemma 1 (Reduction and free variables)
If $t \to_\beta t'$ and x is free in t' then x is free in t.

Instead of the general β-rule, we will be interested in a restriction called the *weak head reduction*: a redex can be reduced only if it is just under some applications. That is, we only reduce redexes of the shape $((\lambda x.t)u)u_1 \dots u_n$. A *weak head normal form* is a normal form for this particular reduction procedure, that is a λ-term of the shape $(x)u_1 \dots u_n$ (in this case x is called its *head variable*) or $\lambda x.t$.

1.2 Simple types

Given a set of *ground types* (σ, ...), the *simple types* are generated by:

$$A \quad ::= \quad \sigma \quad | \quad \perp \quad | \quad A \to A$$

with the convention $A \to B \to C = A \to (B \to C)$. The *atoms* σ or $\perp$ are denoted by X. Any simple type can be written in a unique way in the shape $A = A_1 \to \cdots \to A_n \to X$. The constant $\perp$ can be considered as a particular ground type and will be used later.

A typing context Γ is a finite set of pairs (x, A), denoted by $x : A$, where each λ-variable appears at most once. A typing judgment has the shape $\Gamma \vdash t : A$. The typing rule we are using for the simply typed λ-calculus are:

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

where $\Gamma \setminus \{x : A\}$ is not defined if Γ contains $x : B$ with $A \neq B$, and $\Gamma \cup \Delta$ is not defined if Γ contains $x : A$ and Δ contains $x : B$ with $A \neq B$ (we also use $\Gamma, x : A$ for $\Gamma \cup \{x : A\}$).

The typing system we present doesn't allow to declare unused variables (which is not the case with a *var*-rule like $\Gamma, x : A \vdash x : A$). This requires to give a refined statement for the subject reduction property (proposition 1) but gives more informative typing judgments as stated by lemma 2 or lemma 5. Moreover, this makes the typing system of section 2.2 easier to understand.

Lemma 2 (Typing context and free variables)
If $\Gamma \vdash t : A$ is derivable, then Γ contains exactly one typing declaration for each free variable of t.

Lemma 3 (Substitution)
If $\Gamma, x : B \vdash t : A$ and $\Delta \vdash u : B$ are derivable and $\Gamma \cup \Delta$ is defined, then $\Gamma \cup \Delta \vdash t[{}^{u}/_{x}] : A$.

Proposition 1 (Subject reduction)
If $\Gamma \vdash t : A$ is derivable and $t \to_\beta t'$, then $\Gamma' \vdash t' : A$ is derivable where Γ' is the subset of Γ containing only the typing declarations for the free variables of t'.

Proposition 2 (Strong normalization)
If $\Gamma \vdash t : A$ is derivable, there is no infinite sequence of reductions starting from t.

2 Krivine's abstract machine

Instead of the usual interpretation of the reduction of the λ-calculus as a rewriting system, we will interpret the constructions of the λ-calculus as instructions of an abstract machine: Krivine's abstract machine (or KAM).

2.1 Definitions and properties

In order to define the states of the KAM, we need the following mutually inductive definitions:

- An *environment* e is a partial function with finite domain from λVar to the set of closures (or equivalently a finite set of pairs (x, c)).
- A *closure* c is a pair (t, e) of a λ-term and an environment (at this stage, there is no particular requirement on the fact e must (or not) give values for the free variables of t, some additional constraints will be required by typing in section 2.2, see comment page 8).
- A *stack* π is a finite sequence of closures.
- A *state* is a triple $\langle\, t\ ,\ e\ ,\ \pi \,\rangle$ (or equivalently a pair (c, π)).

Informally, a λ-term requires some information to define the value of its free variables, and this information is given by the environment in a closure. A stack is an evaluation context.

We define the following notations:

- $\emptyset$ is the empty environment;
- if e is an environment, $e(x)$ is the closure associated with x in e;
- $e + (x = c)$ is the environment obtained by modifying the value associated with x in e (or by defining it if $e(x)$ was undefined), which is c in the new environment;
- ε is the empty stack;
- $c :: \pi$ is the stack obtained by pushing c on π.

Expansion of states. We can transform any λ-term t into a state $\langle\, t \,,\, \emptyset \,,\, \varepsilon \,\rangle$ of the machine. The converse is also possible:

- if $c = (t, e)$ is a closure with $e = \{(x_1, c_1), \ldots, (x_n, c_n)\}$, the λ-term $\widetilde{c}$ or $t\{e\}$ is given by the corresponding substitutions $t[{}^{\widetilde{c_1}}/_{x_1}, \ldots, {}^{\widetilde{c_n}}/_{x_n}]$;
- if $\pi = c_1 :: \cdots :: c_n :: \varepsilon$ is a stack, $\widetilde{\pi}$ is the sequence of λ-terms $\widetilde{c_1} \ldots \widetilde{c_n}$.

If $s = \langle\, t \,,\, e \,,\, \pi \,\rangle$ is a state of the machine, the *expansion* of s is $\widetilde{s} = (t\{e\})\widetilde{\pi}$.

Transitions. The transitions of the machine give the evolution of states with the idea that the λ-term in the state is the set of instructions and defines the transition to be applied:

$$\langle\, (t)u \,,\, e \,,\, \pi \,\rangle \xrightarrow{push} \langle\, t \,,\, e \,,\, (u, e) :: \pi \,\rangle$$

$$\langle\, \lambda x.t \,,\, e \,,\, c :: \pi \,\rangle \xrightarrow{pop} \langle\, t \,,\, e + (x = c) \,,\, \pi \,\rangle$$

$$\langle\, x \,,\, e \,,\, \pi \,\rangle \xrightarrow{deref} \langle\, t \,,\, e' \,,\, \pi \,\rangle \qquad \text{where } e(x) = (t, e')$$

If none of the transitions can be applied, the machine stops and the last state is the *result* of the computation. There are two reasons for the machine to stop:

- in a (*pop*) transition if the stack is empty;
- in a (*deref*) transition if the variable is not defined in the environment.

In section 2.2, we will see how we can control these stopping cases by some typing constraints.

Intuitive properties. We first give some intuitions about the properties of the KAM, which will be made more formal in the sequel.

- In a closure (t, e), we can modify in e the value associated with any variable not free in t without modifying computation (lemma 4).
- We can replace any closure (t, e) by $(t\{e\}, \emptyset)$ without modifying computation.
- The computation of the KAM realizes the weak head reduction of λ-terms.

Example 1 (Computation of a β-redex)
The evaluation of a β-redex in the KAM starts by:

$$\begin{array}{rlllll}
 & \langle & (\lambda x.t)u & , & e & , & \pi & \rangle \\
\xrightarrow{push} & \langle & \lambda x.t & , & e & , & (u,e) :: \pi & \rangle \\
\xrightarrow{pop} & \langle & t & , & e + (x = (u,e)) & , & \pi & \rangle
\end{array}$$

and during the evaluation of t, each time the variable x arrives in head position, a (*deref*) transition is used to evaluate u (this corresponds to the substitution of x by u given by the β-reduction).

Lemma 4 (Non-free variables)
If (t,e) is a closure appearing in the state s and if s' is obtained by replacing (t,e) by $(t, e+(x=c))$ in s with x not free in t, the λ-terms that will appear in "instruction position" during the computation of s' in the KAM *are the same as for the computation from s.*

PROOF: We consider the following relation on states: $s \leq s'$ if s' is obtained from s by replacing some closures (t,e) of s by $(t, e+(x=c))$ with x not free in t.

We show that if $s \longrightarrow s_1$ and $s \leq s'$ then $s' \longrightarrow s'_1$ with $s_1 \leq s'_1$. We consider each possible transition for $s \longrightarrow s_1$ with $s = \langle\, t_0 \,,\, e_0 \,,\, \pi_0 \,\rangle$ and $s' = \langle\, t_0 \,,\, e'_0 \,,\, \pi'_0 \,\rangle$:

(*push*) If $t_0 = (u_0)v_0$, then $s_1 = \langle\, u_0 \,,\, e_0 \,,\, (v_0, e_0) :: \pi_0 \,\rangle$ and $s' \longrightarrow s'_1$ with $s'_1 = \langle\, u_0 \,,\, e'_0 \,,\, (v_0, e'_0) :: \pi'_0 \,\rangle$. Since $s \leq s'$, the difference between e_0 and e'_0 can only be some additional declarations for variables not free in $(u_0)v_0$ and the same for π_0 and π'_0 so that $s_1 \leq s'_1$.

(*pop*) If $t_0 = \lambda y.u_0$ then $\pi_0 = c_0 :: \pi_1$ so that $s_1 = \langle\, u_0 \,,\, e_0 + (y = c_0) \,,\, \pi_1 \,\rangle$, $\pi'_0 = c'_0 :: \pi'_1$ and $s'_1 = \langle\, u_0 \,,\, e'_0 + (y = c'_0) \,,\, \pi'_1 \,\rangle$. Up to α-equivalence, we can assume that y is not declared in e_0 and e'_0 and from $s \leq s'$ we easily deduce $s_1 \leq s'_1$.

(*deref*) If $t_0 = y$ and $e_0(y) = (t_1, e_1)$ then $e'_0(y)$ is defined to be some (t'_1, e'_1) and $s_1 = \langle\, t_1 \,,\, e_1 \,,\, \pi_0 \,\rangle$, $s'_1 = \langle\, t'_1 \,,\, e'_1 \,,\, \pi'_0 \,\rangle$. Since y is free in y, the difference between (t_1, e_1) and (t'_1, e'_1) can only concern declarations in e_1 and e'_1 (and moreover for variables not free in t_1 and t'_1) thus $t_1 = t'_1$, and we then get $s_1 \leq s'_1$.

We thus have that if $s \longrightarrow s_1 \longrightarrow \cdots \longrightarrow s_n \longrightarrow \cdots$ is the sequence of states of the execution of the KAM with starting state s then the sequence of states with starting state s' is $s' \longrightarrow s'_1 \longrightarrow \cdots \longrightarrow s'_n \longrightarrow \cdots$ with $s_i \leq s'_i$ for every i. This entails that the term of the state s_i is the same as the term of the state s'_i. $\square$

Example 2 (η-reduction)
The η-reduction of the λ-calculus is defined by:

$$\lambda x.(t)x \quad \rightarrow_\eta \quad t \qquad\qquad \text{if } x \notin t$$

This reduction is realized by the KAM, except if the starting stack is empty:

$$\langle \quad \lambda x.(t)x \quad , \quad e \quad , \quad \varepsilon \quad \rangle$$

is a stopping state (this will be refined by typing, see after corollary 4.1).

If the stack is non-empty, the computation leads to t:

$$\begin{array}{rlcccccl}
 & \langle & \lambda x.(t)x & , & e & , & c :: \pi & \rangle \\
\xrightarrow{pop} & \langle & (t)x & , & e + (x = c) & , & \pi & \rangle \\
\xrightarrow{push} & \langle & t & , & e + (x = c) & , & (x, e + (x = c)) :: \pi & \rangle
\end{array}$$

and the machine computes as for $\langle\, t \;,\; e \;,\; c :: \pi \,\rangle$ by lemma 4. This is very similar to what appends for β-reduction.

Example 3 (Ω)
We use the standard notation $\delta = \lambda x.(x)x$:

$$\begin{array}{rlcccccl}
 & \langle & (\delta)\delta & , & \emptyset & , & \varepsilon & \rangle \\
\xrightarrow{push} & \langle & \delta & , & \emptyset & , & (\delta, \emptyset) & \rangle \\
\xrightarrow{pop} & \langle & (x)x & , & x = (\delta, \emptyset) & , & \varepsilon & \rangle \\
\xrightarrow{push} & \langle & x & , & x = (\delta, \emptyset) & , & (x, x = (\delta, \emptyset)) & \rangle \\
\xrightarrow{deref} & \langle & \delta & , & \emptyset & , & (x, x = (\delta, \emptyset)) & \rangle \\
\xrightarrow{pop} & & & & \dots & & &
\end{array}$$

and the computation does not terminate.

We have described a relation between states and λ-terms through the expansion of states and we can extend this relation to the computational part of the two worlds.

Proposition 3 (Simulation)
If an execution of the machine goes from the state s to the state s', the λ-term $\widetilde{s}$ reduces by weak head reduction to $\widetilde{s'}$. This simulation is strict *in the case of (pop) transitions: if $s \xrightarrow{pop} s'$ then $\widetilde{s} \to_\beta \widetilde{s'}$.*

Proof: We consider each case of transition:

- (*push*) If the starting state is $s = \langle\, (t)u \;,\; e \;,\; \pi \,\rangle$, we have $\widetilde{s} = ((t)u\{e\})\widetilde{\pi} = (t\{e\})u\{e\}\widetilde{\pi}$ which is exactly $\widetilde{s'}$ with $s' = \langle\, t \;,\; e \;,\; (u, e) :: \pi \,\rangle$, no reduction is required.
- (*pop*) If the starting state is $s = \langle\, \lambda x.t \;,\; e \;,\; c :: \pi \,\rangle$, we have $\widetilde{s} = (\lambda x.t\{e\})\widetilde{c}\widetilde{\pi}$ and $s' = \langle\, t \;,\; e + (x = c) \;,\; \pi \,\rangle$ with $\widetilde{s'} = (t\{e + (x = c)\})\widetilde{\pi} = (t\{e\}[^{\widetilde{c}}/_x])\widetilde{\pi}$. This allows to conclude since $(\lambda x.t\{e\})\widetilde{c}\widetilde{\pi}$ reduces by one step of weak head reduction to $(t\{e\}[^{\widetilde{c}}/_x])\widetilde{\pi}$.
- (*deref*) If the starting state is $s = \langle\, x \;,\; e \;,\; \pi \,\rangle$ with $e(x) = (t, e')$, we have $\widetilde{s} = (x\{e\})\widetilde{\pi} = (t\{e'\})\widetilde{\pi}$ and $s' = \langle\, t \;,\; e' \;,\; \pi \,\rangle$ with $\widetilde{s'} = (t\{e'\})\widetilde{\pi}$, and no reduction is required. $\square$

Corollary 3.1 (Weak head normal form)
If the KAM *starts from the state s and stops in the state s', then $\widetilde{s'}$ is the weak head normal form of $\widetilde{s}$.*

Proof: Using proposition 3, we know that $\widetilde{s}$ reduces to $\widetilde{s'}$ by weak head reduction. We just have to show that if the KAM stops in the state s', $\widetilde{s'}$ is a weak head normal form. We have two possible cases:

- $s' = \langle\, \lambda x.u \;,\; e \;,\; \varepsilon \,\rangle$, so that $\widetilde{s'} = \lambda x.u\{e\}$ is a weak head normal form;
- $s' = \langle\, x \;,\; e \;,\; \pi \,\rangle$ with $e(x)$ undefined, so that $\widetilde{s'} = (x\{e\})\widetilde{\pi} = (x)\widetilde{\pi}$ is a weak head normal form. $\square$

In particular, if the starting state is $\langle\, t \;,\; \emptyset \;,\; \varepsilon \,\rangle$ and the stopping state is $\langle\, u \;,\; e \;,\; \pi \,\rangle$, $(u\{e\})\widetilde{\pi}$ is the weak head normal form of t.

2.2 Typing

In order to type the machine, we first introduce *simple stack types*:

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

where σ is a simple ground type and A is a simple type. The notation $\bar{\sigma}$ suggests a kind of duality between simple types and simple stack types which is made explicit in the remark after the typing rules.

To the previously defined typing judgments for λ-terms $\Gamma \vdash t : A$, we add:

- typing judgments for closures: $\Gamma \vdash_{\texttt{clos}} c : A$, where Γ is a typing context for the free variables of c (that is free in the λ-term of c and not defined in the environment of c) and A is a simple type for the λ-term of c;
- typing judgments for environments: $\Gamma \vdash_{\texttt{env}} e : \{\Xi\}$, where Γ is a typing context for the free variables of e (that is free in a closure of e) and Ξ is a typing context for the variables defined in e;
- typing judgments for stacks: $\Gamma \vdash_{\texttt{stack}} \pi : P$, where Γ is a typing context for the free variables of π (that is free in a closure of π) and P is a simple stack type;
- typing judgments for states: $\Gamma \vdash_{\texttt{state}} s$, where Γ is a typing context for the free variables of s (that is free in the closure of s or in the stack of s).

The typing rules for environments, closures, stacks and states are:

$$\frac{}{\vdash_{\texttt{env}} \emptyset : \{\}} \qquad \frac{\Gamma \vdash_{\texttt{env}} e : \{\Xi\} \qquad \Delta \vdash_{\texttt{clos}} c : A}{\Gamma \cup \Delta \vdash_{\texttt{env}} e + (x = c) : \{\Xi, x : A\}} \qquad \frac{\Gamma \vdash t : A \qquad \Delta \vdash_{\texttt{env}} e : \{\Xi\}}{(\Gamma \setminus \Xi) \cup \Delta \vdash_{\texttt{clos}} (t, e) : A}$$

$$\frac{}{\vdash_{\texttt{stack}} \varepsilon : \bar{\sigma}} \qquad \frac{}{\vdash_{\texttt{stack}} \varepsilon : \top} \qquad \frac{\Gamma \vdash_{\texttt{clos}} c : A \qquad \Delta \vdash_{\texttt{stack}} \pi : P}{\Gamma \cup \Delta \vdash_{\texttt{stack}} (c :: \pi) : A \wedge P}$$

$$\frac{\Gamma \vdash_{\texttt{clos}} (t, e) : A_1 \to \cdots \to A_n \to X \qquad \Delta \vdash_{\texttt{stack}} \pi : A_1 \wedge \cdots \wedge A_n \wedge \bar{X}}{\Gamma \cup \Delta \vdash_{\texttt{state}} \langle\, t \;,\; e \;,\; \pi \,\rangle}$$

with $\bar{X} = \bar{\sigma}$ if $X = \sigma$ and $\bar{X} = \top$ if $X = \bot$.

Remark: This last rule can be defined in a slightly different way if we introduce an explicit "duality" between simple types and simple stack types by $\sigma^{\bot} = \bar{\sigma}$, $\bot^{\bot} = \top$ and $(A \to B)^{\bot} = A \wedge B^{\bot}$:

$$\frac{\Gamma \vdash_{\texttt{clos}} (t, e) : A \qquad \Delta \vdash_{\texttt{stack}} \pi : A^{\bot}}{\Gamma \cup \Delta \vdash_{\texttt{state}} \langle\, t \;,\; e \;,\; \pi \,\rangle}$$

This idea of a duality between terms and stacks can be thought of as a key ingredient of the computational interpretations of classical logic. It appears in Parigot's proof of strong normalization for the $\lambda\mu$-calculus [15] and in Krivine's classical realizability [12], it is related to the duality between call-by-name and call-by-value computations [18, 3], it can be related with the duality of linear logic [4], ...

We can now make more formal the intuitive descriptions of the meanings of typing judgments we have given before the rules.

Definition 1 (Free variables)
We extend the notion of *free variable* to environments, closures, stacks and states:

- If $e = \{(x_1, c_1), \ldots, (x_n, c_n)\}$, the free variables of e are the free variables of c_1, ..., c_n.
- If $c = (t, e)$, the free variables of c are the free variables of t not defined in e and the free variables of e.
- If $\pi = c_1 :: \cdots :: c_n :: \varepsilon$, the free variables of π are the free variables of c_1, ..., c_n.
- If $s = \langle\, t\ ,\ e\ ,\ \pi\,\rangle$, the free variables of s are the free variables of the closure (t, e) and the free variables of π.

It is not very natural to consider closures with free variables, since the intuition behind a closure is a *closed* object containing exactly the declarations required for the free variables of the λ-term. In fact these free variables in closures (or states) should be considered as *constants* more than variables and this gives a very easy way to enrich the language with *ground type constants* since they are treated just like variables.

Lemma 5 (Typing judgments and free variables)
Derivable typing judgments declare exactly the free variables:

- *If* $\Gamma \vdash_{\texttt{env}} e : \{\Xi\}$ *is derivable, then* Γ *contains exactly one typing declaration for each free variable of* e *and* Ξ *contains exactly one typing declaration for each variable declared in* e*.*
- *If* $\Gamma \vdash_{\texttt{clos}} c : A$ *is derivable, then* Γ *contains exactly one typing declaration for each free variable of* c*.*
- *If* $\Gamma \vdash_{\texttt{stack}} \pi : P$ *is derivable, then* Γ *contains exactly one typing declaration for each free variable of* π*.*
- *If* $\Gamma \vdash_{\texttt{state}} s$ *is derivable, then* Γ *contains exactly one typing declaration for each free variable of* s*.*

PROOF: We prove all the results together by induction on the size of the typing derivation. We consider each possible case of a last rule:

- If we derive $\vdash_{\texttt{env}} \emptyset : \{\}$, $\emptyset$ contains no declaration and no free variable.
- If we derive $\Gamma \cup \Delta \vdash_{\texttt{env}} e + (x = c) : \{\Xi, x : A\}$ from $\Gamma \vdash_{\texttt{env}} e : \{\Xi\}$ and $\Delta \vdash_{\texttt{clos}} c : A$, the defined variables of $e + (x = c)$ are those of e, contained in Ξ by induction hypothesis, and x. The free variables of $e + (x = c)$ are those of e, contained in Γ by induction hypothesis, and those of c, contained in Δ by induction hypothesis.

- If we derive $(\Gamma \setminus \Xi) \cup \Delta \vdash_{\texttt{clos}} (t, e) : A$ from $\Gamma \vdash t : A$ and $\Delta \vdash_{\texttt{env}} e : \{\Xi\}$, the free variables of (t, e) are the free variables of t not defined in e contained in $\Gamma \setminus \Xi$ and the free variables of e contained in Δ.
- If we derive $\vdash_{\texttt{stack}} \varepsilon : \bar{\sigma}$ or $\vdash_{\texttt{stack}} \varepsilon : \top$, ε has no free variable.
- If we derive $\Gamma \cup \Delta \vdash_{\texttt{stack}} (c :: \pi) : A \wedge P$ from $\Gamma \vdash_{\texttt{clos}} c : A$ and $\Delta \vdash_{\texttt{stack}} \pi : P$, the free variables of $(c :: \pi)$ are the free variables of c contained in Γ and those of π contained in Δ.
- If we derive $\Gamma \cup \Delta \vdash_{\texttt{state}} \langle\, t \ ,\ e \ ,\ \pi \,\rangle$ from $\Gamma \vdash_{\texttt{clos}} (t, e) : A$ and $\Delta \vdash_{\texttt{stack}} \pi : A^{\perp}$, the free variables of $\langle\, t \ ,\ e \ ,\ \pi \,\rangle$ are the free variables of (t, e) contained in Γ and those of π contained in Δ. □

The typing rules we have given for the KAM are compatible with the typing rules for terms through the expansion of states.

Lemma 6 (Typing and expansion)
If $\Gamma \vdash_{\texttt{state}} s$ *is derivable, there exists an atom* X *such that* $\Gamma' \vdash \widetilde{s} : X$ *is derivable where* Γ' *is the subset of* Γ *containing only the typing declarations for the free variables of* $\widetilde{s}$.

Proof: We first prove, by induction on the size of e, that if $\Gamma \vdash_{\texttt{clos}} (t, e) : A$ then $\Gamma' \vdash t\{e\} : A$. From $\Gamma \vdash_{\texttt{clos}} (t, e) : A$ we can deduce $\Gamma = (\Gamma_1 \setminus \Xi) \cup \Gamma_2$ with $\Gamma_1 \vdash t : A$ and $\Gamma_2 \vdash_{\texttt{env}} e : \{\Xi\}$. If $e = \emptyset$, $t\{e\} = t$ and Γ_2 and Ξ are empty so that $\Gamma \vdash t\{e\} : A$. If $e = e' + (x = c)$, we have $\Gamma_2 = \Gamma_2' \cup \Gamma_2''$ and $\Xi = \Xi', x : B$ with $\Gamma_2' \vdash_{\texttt{env}} e' : \{\Xi'\}$ and $\Gamma_2'' \vdash_{\texttt{clos}} c : B$. We can deduce $(\Gamma_1 \setminus \Xi') \cup \Gamma_2' \vdash_{\texttt{clos}} (t, e') : A$ and by induction hypothesis $\Delta \vdash t\{e'\}$ for the correct subset Δ of $(\Gamma_1 \setminus \Xi') \cup \Gamma_2'$. If x is not free in t, this is enough to conclude, otherwise $t\{e\} = t\{e'\}[{}^{\widetilde{c}}/_{x}]$ and by induction hypothesis $\Gamma_2'' \vdash_{\texttt{clos}} c : B$ entails $\Delta' \vdash \widetilde{c} : B$ with $\Delta' \subset \Gamma_2''$ so that $(\Delta \setminus \{x : B\}) \cup \Delta' \vdash t\{e'\}[{}^{\widetilde{c}}/_{x}] : A$ by lemma 3 since Δ contains $x : B$.

If $\Gamma \vdash_{\texttt{state}} s$ is derivable with $s = \langle\, t \ ,\ e \ ,\ \pi \,\rangle$, we have $\Gamma = \Gamma_1 \cup \Gamma_2$ with $\Gamma_1 \vdash_{\texttt{clos}} (t, e) : A_1 \to \cdots \to A_n \to X$ and $\Gamma_2 \vdash_{\texttt{stack}} \pi : A_1 \wedge \cdots \wedge A_n \wedge \bar{X}$, this entails $\pi = c_1 :: \cdots :: c_n :: \varepsilon$ with, for each $1 \leq i \leq n$, $\Delta_i \vdash_{\texttt{clos}} c_i : A_i$ and $\Gamma_2 = \Delta_1 \cup \cdots \cup \Delta_n$. By the result we have just shown, we get $\Gamma_1' \vdash t\{e\} : A_1 \to \cdots \to A_n \to X$ and, for each $1 \leq i \leq n$, $\Delta_i' \vdash \widetilde{c_i} : A_i$, leading to $\Gamma_1' \cup \Delta_1' \cup \cdots \cup \Delta_n' \vdash (t\{e\})\widetilde{c_1} \ldots \widetilde{c_n} : X$ with $(t\{e\})\widetilde{c_1} \ldots \widetilde{c_n} = \widetilde{s}$. □

Proposition 4 (Subject reduction)
The evaluation of the KAM *preserves typing,* i.e. *if* $\Gamma \vdash_{\texttt{state}} \langle\, t \ ,\ e \ ,\ \pi \,\rangle$ *is derivable and if the following transition is valid:*

$$\langle\, t \ ,\ e \ ,\ \pi \,\rangle \quad \longrightarrow \quad \langle\, t' \ ,\ e' \ ,\ \pi' \,\rangle$$

then $\Gamma' \vdash_{\texttt{state}} \langle\, t' \ ,\ e' \ ,\ \pi' \,\rangle$ *where* Γ' *is the subset of* Γ *containing only the typing declarations for the free variables of* $\langle\, t' \ ,\ e' \ ,\ \pi' \,\rangle$.

Proof: If $\Gamma \vdash_{\texttt{state}} \langle\, t \ ,\ e \ ,\ \pi \,\rangle$ is derivable, we must have $\Gamma = (\Gamma_1 \setminus \Xi) \cup \Gamma_2 \cup \Gamma_3$ with $\Gamma_1 \vdash t : A$, $\Gamma_2 \vdash_{\texttt{env}} e : \{\Xi\}$ and $\Gamma_3 \vdash_{\texttt{stack}} \pi : A^{\perp}$. We look at each possible transition:

(*push*) We have $t = (t')u$, this entails $\Gamma_1 = \Gamma_1' \cup \Gamma_1''$ with $\Gamma_1' \vdash t' : B \to A$ and $\Gamma_1'' \vdash u : B$, and we can derive:

$$\dfrac{\dfrac{\Gamma'_1 \vdash t' : B \to A \qquad \Gamma_2 \vdash_{\mathtt{env}} e : \{\Xi\}}{(\Gamma'_1 \setminus \Xi) \cup \Gamma_2 \vdash_{\mathtt{clos}} (t', e) : B \to A} \qquad \dfrac{\dfrac{\Gamma''_1 \vdash u : B \qquad \Gamma_2 \vdash_{\mathtt{env}} e : \{\Xi\}}{(\Gamma''_1 \setminus \Xi) \cup \Gamma_2 \vdash_{\mathtt{clos}} (u, e) : B} \qquad \Gamma_3 \vdash_{\mathtt{stack}} \pi : A^{\perp}}{(\Gamma''_1 \setminus \Xi) \cup \Gamma_2 \cup \Gamma_3 \vdash_{\mathtt{stack}} ((u, e) :: \pi) : B \wedge A^{\perp}}}{(\Gamma'_1 \setminus \Xi) \cup (\Gamma''_1 \setminus \Xi) \cup \Gamma_2 \cup \Gamma_3 \vdash_{\mathtt{state}} \langle\, t' \ , \ e' \ , \ \pi' \,\rangle}$$

with $(\Gamma'_1 \setminus \Xi) \cup (\Gamma''_1 \setminus \Xi) = (\Gamma'_1 \cup \Gamma''_1) \setminus \Xi = \Gamma_1 \setminus \Xi$ and the free variables of s and s' are the same.

(*pop*) We have $t = \lambda x.t'$, $A = B \to C$, $\Gamma_1 = \Gamma'_1 \setminus \{x : B\}$ and $\Gamma'_1 \vdash t' : C$. Moreover $\pi = c :: \pi'$, thus $\Gamma_3 = \Gamma'_3 \cup \Gamma''_3$ with $\Gamma'_3 \vdash_{\mathtt{clos}} c : B$ and $\Gamma''_3 \vdash_{\mathtt{stack}} \pi' : C^{\perp}$ and we can derive:

$$\dfrac{\dfrac{\Gamma'_1 \vdash t' : C \qquad \dfrac{\Gamma_2 \vdash_{\mathtt{env}} e : \{\Xi\} \qquad \Gamma'_3 \vdash_{\mathtt{clos}} c : B}{\Gamma_2 \cup \Gamma'_3 \vdash_{\mathtt{env}} e + (x = c) : \{\Xi, x : B\}}}{(\Gamma_1 \setminus \Xi) \cup \Gamma_2 \cup \Gamma'_3 \vdash_{\mathtt{clos}} (t', e + (x = c)) : C} \qquad \Gamma''_3 \vdash_{\mathtt{stack}} \pi' : C^{\perp}}{(\Gamma_1 \setminus \Xi) \cup \Gamma_2 \cup \Gamma'_3 \cup \Gamma''_3 \vdash_{\mathtt{state}} \langle\, t' \ , \ e' \ , \ \pi' \,\rangle}$$

and the free variables of s and s' are the same.

(*deref*) We have $t = x$, this entails $\Gamma_1 = x : A$ and $e = e_0 + (x = (t', e'))$. From $\Gamma_2 \vdash_{\mathtt{env}} e_0 + (x = (t', e')) : \{\Xi\}$, we deduce $\Gamma_2 = \Gamma'_2 \cup \Gamma''_2$ and $\Xi = \Xi', x : A$ with $\Gamma'_2 \vdash_{\mathtt{env}} e_0 : \{\Xi'\}$ and $\Gamma''_2 \vdash_{\mathtt{clos}} (t', e') : A$, and we can derive:

$$\dfrac{\Gamma''_2 \vdash_{\mathtt{clos}} (t', e') : A \qquad \Gamma_3 \vdash_{\mathtt{stack}} \pi : A^{\perp}}{\Gamma''_2 \cup \Gamma_3 \vdash_{\mathtt{state}} \langle\, t' \ , \ e' \ , \ \pi' \,\rangle}$$

By lemma 5, $\Gamma''_2 \cup \Gamma_3$ contains exactly the typing declarations from Γ for the free variables of $\langle\, t' \ , \ e' \ , \ \pi' \,\rangle$. □

Corollary 4.1 (Termination)
If $\Gamma \vdash_{\mathtt{state}} s$ is derivable, the KAM*, starting from the state s, stops in a state $\langle\, x \ , \ e' \ , \ \pi' \,\rangle$ where x is the head variable of the weak head normal form of $\widetilde{s}$.*

Proof: We first show that the machine stops, which means that we can't have an infinite sequence of transitions from s. We define the size $|t|$ of a term t to be its number of symbols, the size $|e|$ of an environment e to be the sum of the sizes of its closures and the size of a closure (t, e) to be $|t| + |e|$. We remark that we can't have an infinite sequence of transitions (*push*) and (*deref*) because the value of $(|e|, |t|)$ (ordered in lexicographic order) is strictly decreasing in such a sequence. So that an infinite sequence of transitions of the machine contains an infinite number of transitions (*pop*). According to proposition 3, each (*pop*) transition from s_1 to s_2 corresponds to a step of weak head reduction from $\widetilde{s_1}$ to $\widetilde{s_2}$, but by lemma 6 the term $\widetilde{s}$ associated with the first state is typable so that, by strong normalization for the simply typed λ-calculus (proposition 2), it can't have an infinite sequence of reductions.

Let s' be the stopping state, it is either $\langle\, \lambda x.u \ , \ e' \ , \ \varepsilon \,\rangle$ or $\langle\, x \ , \ e' \ , \ \pi' \,\rangle$ (see page 4) but the first case is rejected by proposition 4 since it must be typable and ε is not typable of a type $A \wedge P$. This entails, $s' = \langle\, x \ , \ e' \ , \ \pi' \,\rangle$ with x not declared in e' and by corollary 3.1, $\widetilde{(x\{e'\})\pi'} = \widetilde{(x)\pi'}$ is the weak head normal form of $\widetilde{s}$ so that x is the head variable of this weak head normal form. □

In particular, the machine never stops because the λ-term is $\lambda x.t$ and the stack is empty (in example 2, the stack cannot be empty).

A natural particular case of the previous corollary is $s = \langle\, t \,,\, \emptyset \,,\, \varepsilon \,\rangle$, and we can wonder what result we get if t is closed and $\vdash_{\texttt{state}} \langle\, t \,,\, \emptyset \,,\, \varepsilon \,\rangle$. In fact, we get a contradiction! If $\vdash_{\texttt{state}} \langle\, t \,,\, \emptyset \,,\, \varepsilon \,\rangle$ is derivable, the KAM ends with a state $\langle\, x \,,\, e' \,,\, \pi' \,\rangle$, but this is not possible because x cannot be defined in e' (otherwise the machine doesn't stop) and for $\vdash_{\texttt{state}} \langle\, x \,,\, e' \,,\, \pi' \,\rangle$ to be derivable, x must be defined in e'. We can give a direct proof that $\vdash_{\texttt{state}} \langle\, t \,,\, \emptyset \,,\, \varepsilon \,\rangle$ is never derivable: if it is, the last rules have the shape:

$$\dfrac{\dfrac{\vdash t : A \qquad \vdash_{\texttt{env}} \emptyset : \{\}}{\vdash_{\texttt{clos}} (t, \emptyset) : A} \qquad \vdash_{\texttt{stack}} \varepsilon : A^{\perp}}{\vdash_{\texttt{state}} \langle\, t \,,\, \emptyset \,,\, \varepsilon \,\rangle}$$

This entails $A = \sigma$ or $A = \perp$, otherwise $\vdash_{\texttt{stack}} \varepsilon : A^{\perp}$ is not derivable, but in this case $\vdash t : A$ is not derivable.

There are two interpretations of this remark. First, using our discussion about free variables in closures and states, we can restrict ourselves to λ-terms containing constants (with ground types) and with a reduction leading to such a constant. Second, we can liberalize the typing rule for ε by:

$$\dfrac{}{\vdash_{\texttt{stack}} \varepsilon : P}$$

with any simple stack type P, but in this case we get back the stopping configuration of a (*pop*) transition that we cannot apply because the stack is empty. So that typing doesn't give anymore a control on the stopping configurations.

3 KAM and control

3.1 Extension of the machine

If we look at a state of the KAM as a pair of a closure and a stack, the transitions we have seen interpret the instructions given by λ-terms as operations on closures: stocking a closure on the stack (*push*), naming a closure in the environment (*pop*) and reading back a closure from its name in the environment (*deref*).

In this spirit, it is natural to try to extend our set of instructions (thus our constructions of terms) in order to define operations on stacks: naming a stack in the environment (*save*) and reading back a stack from its name in the environment (*restore*).

Given a denumerable set μVar of variables, denoted by α, β, ..., called the μ-variables and used in the machine as names for the stacks, we extend the language of terms with two new constructions:

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

The μ-construction is a binder for the μ-variable α and $[\alpha]t$ introduces a free occurrence of α. This leads to the following definition of the *free μ-variables* of a term t:

- x doesn't contain any free μ-variable;
- the free μ-variables of $\lambda x.t$ are those of t;
- the free μ-variables of $(t)u$ are those of t and u;
- the free μ-variables of $\mu\alpha.t$ are those of t except α;

- the free μ-variables of $[\alpha]t$ are those of t together with α.

Terms are used up to α-equivalence for both bound λ-variables and bound μ-variables.

An environment will now be a pair of partial functions with finite domain from λVar to closures and from μVar to stacks (or equivalently a finite set of pairs (x, c) and (α, π)).

This allows to extend the KAM with the two new transitions:

$$\langle\, \mu\alpha.t \;,\; e \;,\; \pi \,\rangle \xrightarrow{save} \langle\, t \;,\; e + (\alpha = \pi) \;,\; \varepsilon \,\rangle$$

$$\langle\, [\alpha]t \;,\; e \;,\; \varepsilon \,\rangle \xrightarrow{restore} \langle\, t \;,\; e \;,\; \pi \,\rangle \qquad \text{where } e(\alpha) = \pi$$

These transitions add two new stopping cases for the KAM:

- in a (*restore*) transition if the stack is not empty;
- in a (*restore*) transition if the variable is not defined in the environment.

It would be possible to define the (*restore*) transition with a non-empty starting stack by just discarding it. However this generalized transition can be simulated by replacing $[\alpha]t$ with $\mu\delta[\alpha]t$ ($\delta \notin t$) and our transitions appear as more atomic operations.

Lemma 7 (Non-free variables)
If (t, e) is a closure appearing in the state s and if s' is obtained by replacing (t, e) by $(t, e + (\alpha = \pi))$ in s with α not free in t, the λ-terms that will appear in "instruction position" during the computation of s' in the KAM are the same as for the computation from s.

PROOF: We follow the proof of lemma 4 with the same notations, and we extend the relation $s \leq s'$ by allowing new declarations for non-free μ-variables. Since the transitions (*push*), (*pop*) and (*deref*) ignore the definitions of μ-variables in the environment, we just study the two remaining transitions:

(*save*) If $t_0 = \mu\beta.u_0$ (we assume β not in e_0 and e'_0 using α-equivalence) then $s_1 = \langle\, u_0 \;,\; e_0 + (\beta = \pi_0) \;,\; \varepsilon \,\rangle$ and $s' \xrightarrow{\qquad} s'_1$ with $s'_1 = \langle\, u_0 \;,\; e'_0 + (\beta = \pi'_0) \;,\; \varepsilon \,\rangle$ so that $s_1 \leq s'_1$.

(*restore*) If $t_0 = [\beta]u_0$ and $e_0(\beta) = \pi_1$ then $s_1 = \langle\, u_0 \;,\; e_0 \;,\; \pi_1 \,\rangle$ and $e'_0(\beta)$ is defined because β is free in $[\beta]u_0$ and $s \leq s'$ so that $s' \xrightarrow{\qquad} s'_1$ with $s'_1 = \langle\, u_0 \;,\; e'_0 \;,\; \pi'_1 \,\rangle$ (where $e'_0(\beta) = \pi'_1$). Since β is free in $[\beta]u_0$, we easily verify that $s_1 \leq s'_1$.

We conclude as for lemma 4. □

Example 4 (*push*/*save*)
As shown in example 1, the β-reduction corresponds, in the KAM, to an interaction between the (*push*) instruction which adds a closure on the stack and the (*pop*) instruction which reads a closure from the stack. We now have a new instruction which reads the stack: the (*save*) instruction, and we are going to look at the interaction between (*push*) and (*save*).

$$\begin{array}{rlcccccl}
 & \langle & (\mu\alpha.t)u & , & e & , & \varepsilon & \rangle \\
\xrightarrow{push} & \langle & \mu\alpha.t & , & e & , & (u,e) :: \varepsilon & \rangle \\
\xrightarrow{save} & \langle & t & , & e + (\alpha = (u,e) :: \varepsilon) & , & \varepsilon & \rangle
\end{array}$$

We denote by $\vec{u}$ the sequence $u_1 \dots u_n$ and by $\pi_{\vec{u}}$ the stack $(u_1, e) :: \cdots :: (u_n, e) :: \varepsilon$. Since (*save*) acts on the whole stack, we have the more general behavior:

$$\begin{array}{rlcccccl}
 & \langle & (\mu\alpha.t)\vec{u} & , & e & , & \varepsilon & \rangle \\
(\xrightarrow{push})^n & \langle & \mu\alpha.t & , & e & , & \pi_{\vec{u}} & \rangle \\
\xrightarrow{save} & \langle & t & , & e + (\alpha = \pi_{\vec{u}}) & , & \varepsilon & \rangle
\end{array}$$

but this example is not local in the following sense: it requires an empty starting stack and it looks at an arbitrarily long sequence of terms in the application. The local case would correspond to the starting state $\langle\, (\mu\alpha.t)u \ , \ e \ , \ \pi \,\rangle$:

$$\begin{array}{rlcccccl}
 & \langle & (\mu\alpha.t)u & , & e & , & \pi & \rangle \\
\xrightarrow{push} & \langle & \mu\alpha.t & , & e & , & (u, e) :: \pi & \rangle \\
\xrightarrow{save} & \langle & t & , & e + (\alpha = (u, e) :: \pi) & , & \varepsilon & \rangle
\end{array}$$

As for the β-reduction, we want to find a term t' with the same behavior as $(\mu\alpha.t)u$ to define a reduction rule. If we continue the execution of the KAM described just above for $\langle\, (\mu\alpha.t)u \ , \ e \ , \ \pi \,\rangle$, each time we arrive to some term $[\alpha]v$, we have to restore the stack $(u, e) :: \pi$ which corresponds to the evaluation of $(v)u$ with the stack π, that is to the evaluation of $[\alpha](v)u$ where $\alpha = \pi$ is in the environment.

Consider t' to be $\mu\alpha.t[{}^{[\alpha](v)u}/_{[\alpha]v}]$ where $t[{}^{[\alpha](v)u}/_{[\alpha]v}]$ is obtained by substituting any sub-term of t starting by $[\alpha]$, thus of the shape $[\alpha]v$ for some v, by the term $[\alpha](v)u$ (see section 4.1 for a formal definition of this notion of substitution). We have:

$$\begin{array}{rlcccccl}
 & \langle & \mu\alpha.t[{}^{[\alpha](v)u}/_{[\alpha]v}] & , & e & , & \pi & \rangle \\
\xrightarrow{save} & \langle & t[{}^{[\alpha](v)u}/_{[\alpha]v}] & , & e + (\alpha = \pi) & , & \varepsilon & \rangle
\end{array}$$

According to the previous discussion, the evaluations of $\langle t \ , \ e + (\alpha = (u, e) :: \pi) \ , \ \varepsilon \rangle$ and $\langle\, t[{}^{[\alpha](v)u}/_{[\alpha]v}] \ , \ e + (\alpha = \pi) \ , \ \varepsilon \,\rangle$ are almost the same: in the first case, each time we have a $[\alpha]v$, we evaluate v with the stack $(u, e) :: \pi$ and in the second case, each time we have a $[\alpha]v$, it has been substituted by $[\alpha](v)u$ and we evaluate $(v)u$ with the stack π, that is v with a stack $(u, e') :: \pi$.

Example 5 (*save/restore*)
Putting together the two instructions $\mu\alpha$ and $[\alpha]$ continues the computation in the same conditions except that the stack is memorized in the environment with name α:

$$\begin{array}{rlcccccl}
 & \langle & \mu\alpha[\alpha]t & , & e & , & \pi & \rangle \\
\xrightarrow{save} & \langle & [\alpha]t & , & e + (\alpha = \pi) & , & \varepsilon & \rangle \\
\xrightarrow{restore} & \langle & t & , & e + (\alpha = \pi) & , & \pi & \rangle
\end{array}$$

3.2 Control primitives

To establish a relation between the stack manipulations in the KAM and control operators, we will show how to simulate an extension of the λ-calculus with "jumping" primitives in the extended KAM.

We consider the following toy extension of the λ-calculus:

$$t \quad ::= \quad x \quad | \quad \lambda x.t \quad | \quad (t)t \quad | \quad \texttt{label}\, k : t \quad | \quad \texttt{goto}\, k\, \texttt{with}\, t$$

Since we don't want to give the detailed operational semantics of this language and we just want to be informal in this section, the reader can have a look at [6, 20] for more formal presentations of this kind of extensions of the λ-calculus with control primitives.

The idea is the following: if we want to compute $\texttt{label}\ k : t$ in a context $\mathcal{C}$, we start the computation of t in the context $\mathcal{C}$, and if we arrive to some $\texttt{goto}\ k\ \texttt{with}\ u$, we stop the execution and we start the computation of u in the context $\mathcal{C}$.

We define an embedding of this language in the language of instructions of the KAM:

$$\begin{array}{rcl}
\overline{x} & = & x \\
\overline{\lambda x.t} & = & \lambda x.\overline{t} \\
\overline{(t)u} & = & (\overline{t})\overline{u} \\
\overline{\texttt{label}\, k : t} & = & (\lambda f.\mu\alpha[\alpha](f)\lambda x.\mu\delta[\alpha]x)\lambda k.\overline{t} \\
\overline{\texttt{goto}\, k\, \texttt{with}\, u} & = & (k)\overline{u}
\end{array}$$

We can compare the execution of $\texttt{label}\ k : t$ with the execution of the KAM for the corresponding term $\overline{\texttt{label}\ k : t}$ (this very particular case where we use a unique label defined at top level, can be encoded with both exceptions and continuations even if they usually differ [13, 16]).

$$\begin{array}{rlclclcl}
 & \langle & (\lambda f.\mu\alpha[\alpha](f)\lambda x.\mu\delta[\alpha]x)\lambda k.\overline{t} & , & e & , & \pi & \rangle \\
\xrightarrow{push} & \langle & \lambda f.\mu\alpha[\alpha](f)\lambda x.\mu\delta[\alpha]x & , & e & , & (\lambda k.\overline{t}, e) :: \pi & \rangle \\
\xrightarrow{pop} & \langle & \mu\alpha[\alpha](f)\lambda x.\mu\delta[\alpha]x & , & e + (f = (\lambda k.\overline{t}, e)) & , & \pi & \rangle \\
\xrightarrow{save} & \langle & [\alpha](f)\lambda x.\mu\delta[\alpha]x & , & e + (f = (\lambda k.\overline{t}, e)) + (\alpha = \pi) & , & \varepsilon & \rangle \\
\xrightarrow{restore} & \langle & (f)\lambda x.\mu\delta[\alpha]x & , & e + (f = (\lambda k.\overline{t}, e)) + (\alpha = \pi) & , & \pi & \rangle \\
\xrightarrow{push} & \langle & f & , & e + (f = (\lambda k.\overline{t}, e)) + (\alpha = \pi) & , & (\lambda x.\mu\delta[\alpha]x, e') :: \pi & \rangle \\
\xrightarrow{deref} & \langle & \lambda k.\overline{t} & , & e & , & (\lambda x.\mu\delta[\alpha]x, e') :: \pi & \rangle \\
\xrightarrow{pop} & \langle & \overline{t} & , & e + (k = (\lambda x.\mu\delta[\alpha]x, e')) & , & \pi & \rangle
\end{array}$$

where $e' = e + (f = (\lambda k.\overline{t}, e)) + (\alpha = \pi)$. According to lemmas 4 and 7, we can replace $(\lambda x.\mu\delta[\alpha]x, e')$ with $(\lambda x.\mu\delta[\alpha]x, \alpha = \pi)$.

Using the idea that the stack represents the current context of evaluation, this shows that in order to compute $\overline{\texttt{label}\ k : t}$, we compute $\overline{t}$ in the same context (with some appropriate upgrade of the environment). If k never occurs during the computation of $\overline{t}$, the computations of $\overline{\texttt{label}\ k : t}$ and $\overline{t}$ are the same, this corresponds to the case where no jump to k is used in t. If k appears during the computation of $\overline{t}$, the instruction $(k)\overline{u}$ appears in the KAM:

$$\begin{array}{rlclclcl}
 & \langle & (k)\overline{u} & , & e_0 & , & \pi_0 & \rangle \\
\xrightarrow{push} & \langle & k & , & e_0 & , & (\overline{u}, e_0) :: \pi_0 & \rangle \\
\xrightarrow{deref} & \langle & \lambda x.\mu\delta[\alpha]x & , & \alpha = \pi & , & (\overline{u}, e_0) :: \pi_0 & \rangle \\
\xrightarrow{pop} & \langle & \mu\delta[\alpha]x & , & (\alpha = \pi) + (x = (\overline{u}, e_0)) & , & \pi_0 & \rangle \\
\xrightarrow{save} & \langle & [\alpha]x & , & (\alpha = \pi) + (x = (\overline{u}, e_0)) + (\delta = \pi_0) & , & \varepsilon & \rangle \\
\xrightarrow{restore} & \langle & x & , & (\alpha = \pi) + (x = (\overline{u}, e_0)) + (\delta = \pi_0) & , & \pi & \rangle \\
\xrightarrow{deref} & \langle & \overline{u} & , & e_0 & , & \pi & \rangle
\end{array}$$

which means that we stop the evaluation of $\overline{t}$, and we start the execution of $\overline{u}$ in the *initial context* π as for `label` $k : t$.

4 The $\lambda\mu$-calculus

We are going to move from the KAM to a term language with rewriting, based on the instruction language of the machine with $\mu\alpha$ and $[\alpha]$. This language is a very small variant of M. Parigot's *$\lambda\mu$-calculus* [14] (as given in [9]).

4.1 The language

We have already defined the term language in section 3.1 with its two binders λ for the λ-variables and μ for the μ-variables. We now give the corresponding reduction rules, extending β-reduction with a rule for the μ binder (as suggested in example 4):

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

The substitution $t[{}^{[\alpha](v)u}/{}_{[\alpha]v}]$ is one of the key ingredients of the $\lambda\mu$-calculus, some intuitions have been given in example 4 and it is formally defined as follows:

$$\begin{array}{rll}
x[{}^{[\alpha](v)u}/{}_{[\alpha]v}] = x & & \\
(\lambda x.t)[{}^{[\alpha](v)u}/{}_{[\alpha]v}] = \lambda x.(t[{}^{[\alpha](v)u}/{}_{[\alpha]v}]) & & \text{with } x \notin u \text{ using } \alpha\text{-equivalence} \\
((t)t')[{}^{[\alpha](v)u}/{}_{[\alpha]v}] = (t[{}^{[\alpha](v)u}/{}_{[\alpha]v}])t'[{}^{[\alpha](v)u}/{}_{[\alpha]v}] & & \\
(\mu\beta.t)[{}^{[\alpha](v)u}/{}_{[\alpha]v}] = \mu\beta.(t[{}^{[\alpha](v)u}/{}_{[\alpha]v}]) & & \text{with } \beta \neq \alpha \text{ and } \beta \notin u \text{ using } \alpha\text{-equivalence} \\
([\alpha]t)[{}^{[\alpha](v)u}/{}_{[\alpha]v}] = [\alpha](t[{}^{[\alpha](v)u}/{}_{[\alpha]v}])u & & \\
([\beta]t)[{}^{[\alpha](v)u}/{}_{[\alpha]v}] = [\beta](t[{}^{[\alpha](v)u}/{}_{[\alpha]v}]) & & \text{if } \beta \neq \alpha
\end{array}$$

Example 6 (μ-substitution)
With the definition of the new substitution:

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

In the spirit of η-reduction, we can also add two other reduction rules:

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

We will use the notation $u \to v$ for $u \to_\beta v$ or $u \to_\mu v$ or $u \to_\rho v$.

Proposition 5 (Church-Rosser property)
If $t \to^* u$ and $t \to^* v$, there exists a $\lambda\mu$-term w such that $u \to^* w$ and $v \to^* w$.

Proof: See [14] slightly corrected in [17] for example. □

Example 4 shows how the KAM simulates the μ-reduction: the difference between $\langle\, t\,,\, e + (\alpha = (u, e) :: \pi)\,,\, \varepsilon\,\rangle$ and $\langle\, t[^{[\alpha](v)u}/_{[\alpha]v}]\,,\, e + (\alpha = \pi)\,,\, \varepsilon\,\rangle$ occurs in the execution when some $[\alpha]v$ appears as the current instruction. In the first case, the stack $(u, e) :: \pi$ is restored and execution continues with v. In the second case, we have in fact $[\alpha](v)u$ and the stack π is restored but after one transition u is pushed on the stack and execution continues with v leading to the same computation.

Example 7 (ρ-reduction)
The KAM simulates the ρ-reduction if the starting state has an empty stack and contains a declaration for the variable β in the environment:

$$\begin{array}{rcccccccl} & \langle & [\beta]\mu\alpha.t & , & e + (\beta = \pi) & , & \varepsilon & \rangle \\ \xrightarrow{\textit{restore}} & \langle & \mu\alpha.t & , & e + (\beta = \pi) & , & \pi & \rangle \\ \xrightarrow{\textit{save}} & \langle & t & , & e + (\beta = \pi) + (\alpha = \pi) & , & \varepsilon & \rangle \end{array}$$

Example 8 (θ-reduction)
The simulation of the θ-reduction corresponds to the particular case of example 5 where $\alpha \notin t$:

$$\begin{array}{rcccccccl} & \langle & \mu\alpha[\alpha]t & , & e & , & \pi & \rangle \\ \xrightarrow{\textit{save}} & \langle & [\alpha]t & , & e + (\alpha = \pi) & , & \varepsilon & \rangle \\ \xrightarrow{\textit{restore}} & \langle & t & , & e + (\alpha = \pi) & , & \pi & \rangle \end{array}$$

and, according to lemma 7, the computation follows on like with the state $\langle\, t\,,\, e\,,\, \pi\,\rangle$.

Due to the modification of the definition of environments, we have to extend the notion of *expansion of states*:

- if $c = (t, e)$ is a closure with $e = \{(x_1, c_1), \ldots, (x_n, c_n), (\alpha_1, \pi_1), \ldots, (\alpha_m, \pi_m)\}$, the $\lambda\mu$-term $\widetilde{c}$ or $t\{e\}$ is $t[^{\widetilde{c_1}}/_{x_1}, \ldots, ^{\widetilde{c_n}}/_{x_n}][^{[\alpha_1](v)\widetilde{\pi_1}}/_{[\alpha_1]v}, \ldots, ^{[\alpha_m](v)\widetilde{\pi_m}}/_{[\alpha_m]v}]$.

The *weak head reduction* of the $\lambda\mu$-calculus allows to reduce redexes (for the β-, μ- or ρ-reductions) under applications but also under some $\mu\alpha$ constructions at the beginning of the term followed by some $[\alpha]$ with α bound. This means that the redex r can be reduced in t only if t has the shape:

$$\mu\alpha_1 \ldots \mu\alpha_n[\beta_1] \ldots [\beta_k](r)u_1 \ldots u_p$$

with $n, k, p \geq 0$ and $\beta_j \in \{\alpha_1, \ldots, \alpha_n\}$ for $1 \leq j \leq k$.

A *weak head normal form* is a normal form for this reduction procedure, that is a $\lambda\mu$-term of one of the following shapes:

- $\mu\alpha_1 \dots \mu\alpha_n[\beta_1]\dots[\beta_k][\gamma]t$ with $n \geq 0$, $k \geq 0$, β_1, ..., β_k bound and γ free, in this case γ is called the *head μ-variable* of the weak head normal form;
- $\mu\alpha_1 \dots \mu\alpha_n[\beta_1]\dots[\beta_k]([\gamma]t)u_1 \dots u_p$ with $n \geq 0$, $k \geq 0$, $p > 0$ and β_1, ..., β_k bound, in this case γ is called the *head μ-variable*;
- $\mu\alpha_1 \dots \mu\alpha_n[\beta_1]\dots[\beta_k](x)u_1 \dots u_p$ with $n \geq 0$, $k \geq 0$, $p \geq 0$ and β_1, ..., β_k bound, in this case x is called the *head λ-variable*;
- $\mu\alpha_1 \dots \mu\alpha_n[\beta_1]\dots[\beta_k]\lambda x.t$ with $n \geq 0$, $k \geq 0$ and β_1, ..., β_k bound, and we don't define head variables in this case.

Proposition 6 (Simulation)
If an execution of the machine goes from the state s to the state s', the $\lambda\mu$-term $\widetilde{s}$ reduces by weak head reduction to a term u such that $\widetilde{s'}$ is obtained by removing some $\mu\alpha$ and $[\alpha]$ in the beginning of u. This simulation is strict *in the case of (save) transitions with a non-empty stack: if $s \xrightarrow{save} s'$ and the stack of s is not empty then $\widetilde{s} \to_\mu u$.*

Proof: We look at the two new transitions:

(*save*) If the starting state is $s = \langle\, \mu\alpha.t \;,\; e \;,\; \pi \,\rangle$, we have $\widetilde{s} = (\mu\alpha.t\{e\})\widetilde{\pi}$ which reduces to $\mu\alpha.t\{e\}[^{[\alpha](v)\widetilde{\pi}}/_{[\alpha]v}]$ (by n steps if π contains n closures) and $s' = \langle\, t \;,\; e + (\alpha = \pi) \;,\; \varepsilon \,\rangle$ with $\widetilde{s'} = t\{e + (\alpha = \pi)\} = t\{e\}[^{[\alpha](v)\widetilde{\pi}}/_{[\alpha]v}]$, and the difference is a $\mu\alpha$ in the beginning.

(*restore*) If the starting state is $s = \langle\, [\alpha]t \;,\; e \;,\; \varepsilon \,\rangle$ with $e(\alpha) = \pi$, we have $\widetilde{s} = ([\alpha]t)\{e\} = [\alpha](t\{e\})\widetilde{\pi}$ and $s' = \langle\, t \;,\; e \;,\; \pi \,\rangle$ with $\widetilde{s'} = (t\{e\})\widetilde{\pi}$, and the difference is a $[\alpha]$ in the beginning. □

Corollary 6.1 (Weak head normal form)
If the KAM *starts from the state s and stops in the state s', then $\widetilde{s'}$ is the weak head normal form of $\widetilde{s}$ (up to some $\mu\alpha$ and $[\alpha]$ in the beginning).*

Proof: Using proposition 6, we know that $\widetilde{s}$ reduces to $\widetilde{s'}$ by weak head reduction (up to some $\mu\alpha$ and $[\alpha]$ in the beginning). We just have to show that if the KAM stops in the state s', $\widetilde{s'}$ is a weak head normal form. We have two new possible cases with respect to corollary 3.1:

- $s' = \langle\, [\alpha]u \;,\; e \;,\; \pi \,\rangle$ with $\pi \neq \varepsilon$, so that $\widetilde{s'} = ([\alpha]u')\widetilde{\pi}$ (for some u') is a weak head normal form;
- $s' = \langle\, [\alpha]u \;,\; e \;,\; \varepsilon \,\rangle$ with $e(\alpha)$ undefined, so that $\widetilde{s'} = [\alpha](u\{e\})$ is also a weak head normal form. □

4.2 Simple types

Our goal is to build a typing system for the $\lambda\mu$-calculus out of the KAM, starting with an intuitive typing of the machine. We look at the following sequence:

$$\begin{array}{rllll}
 & \langle\; \mu\beta[\alpha]t & , \quad \alpha = \pi' & , \; \pi & \rangle \\
\xrightarrow{save} & \langle\; [\alpha]t & , \quad (\alpha = \pi') + (\beta = \pi) & , \; \varepsilon & \rangle \\
\xrightarrow{restore} & \langle\; t & , \quad (\alpha = \pi') + (\beta = \pi) & , \; \pi' & \rangle
\end{array}$$

Let A be the type of t, the last state tells us that π' must have type $A^{\perp}$. To ensure the coherence of the environment, the type of α must be in correspondence with the type of π', and if we want to type μ-variables with simple types, the only natural candidate is A. The middle state entails that $[\alpha]t$ must have an atomic type since the corresponding stack is empty, we make the particular choice of $\perp$ for this purpose, so that if t has type A and α has type A, $[\alpha]t$ has type $\perp$, this can be summarized by the informal judgments:

$$\dfrac{t : A}{[\alpha]t : \perp \text{ and } \alpha : A}\;[.]$$

the typing declaration for α is required in the second judgment since α is free in $[\alpha]t$.

Let B be the type of β, according to the middle state, π has type $B^{\perp}$, this entails in the first state that $\mu\beta[\alpha]t$ must have type B. We summarize it by:

$$\dfrac{t : \perp \text{ and } \beta : B}{\mu\beta.t : B}\;mu$$

If we try to compare these informal rules with the (lam) rule written in the same way:

$$\dfrac{x : A \text{ and } t : B}{\lambda x.t : A \to B}\;lam$$

we can see that the type of x appears negatively in the type of t which justifies logically the fact that the typing declaration for x in a typing judgment of the λ-calculus appears on the left-hand side of $\vdash$. Whereas the types of the μ-variables in our informal rules appear in positive occurrence in the type of the term, this leads us to put typing declarations for μ-variables in the right-hand side of the $\vdash$, and to introduce typing judgments of the shape:

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

The formal typing rules for $\mu\alpha$ and $[\alpha]$ follow from these remarks:

$$\dfrac{\Gamma \vdash t : A \mid \Delta}{\Gamma \vdash [\alpha]t : \perp \mid \Delta \cup \{\alpha : A\}}\;[.] \qquad \dfrac{\Gamma \vdash t : \perp \mid \Delta}{\Gamma \vdash \mu\alpha.t : A \mid \Delta \setminus \{\alpha : A\}}\;mu$$

Due to these rules, the atom $\perp$ has now a different status from σ, ...

Lemma 8 (Typing context and free variables)
If $\Gamma \vdash t : A \mid \Delta$ is derivable, then Γ contains exactly one typing declaration for each free λ-variable of t and Δ contains exactly one typing declaration for each free μ-variable of t.

Example 9 (Call/cc and Peirce's law)
The $\lambda\mu$-term we have studied in section 3.2 is typable of type $((A \to B) \to A) \to A$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{f : (A \to B) \to A \vdash f : (A \to B) \to A \mid}\;var \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{x : A \vdash x : A \mid}\;var}{x : A \vdash [\alpha]x : \perp \mid \alpha : A}\;[.]}{x : A \vdash \mu\delta[\alpha]x : B \mid \alpha : A}\;mu}{\vdash \lambda x.\mu\delta[\alpha]x : A \to B \mid \alpha : A}\;lam}{f : (A \to B) \to A \vdash (f)\lambda x.\mu\delta[\alpha]x : A \mid \alpha : A}\;app}{f : (A \to B) \to A \vdash [\alpha](f)\lambda x.\mu\delta[\alpha]x : \perp \mid \alpha : A}\;[.]}{f : (A \to B) \to A \vdash \mu\alpha[\alpha](f)\lambda x.\mu\delta[\alpha]x : A \mid}\;mu}{\vdash \lambda f.\mu\alpha[\alpha](f)\lambda x.\mu\delta[\alpha]x : ((A \to B) \to A) \to A \mid}\;lam$$

The Curry-Howard correspondence gives the relation between typing derivations in the simply typed λ-calculus and intuitionistic logic. T. Griffin [8] has discovered that control operators in programming languages can be typed with classical (not intuitionistically provable) formulas, allowing to extend the Curry-Howard correspondence to classical logic. A lot of work followed this idea in particular the introduction of the $\lambda\mu$-calculus [14] (but also [6, 7, 4]). The λ-terms correspond to proofs in intuitionistic natural deduction and the previous derivation shows that $\lambda\mu$-terms correspond to derivations in classical logic (since intuitionistic logic with Peirce's law is classical logic).

Lemma 9 (Substitution)
If $\Gamma \vdash t : A \mid \Delta, \alpha : B \to C$ *and* $\Gamma' \vdash u : B \mid \Delta'$ *are derivable and* $\Gamma \cup \Gamma'$ *and* $\Delta \cup \Delta'$ *are defined, then* $\Gamma \cup \Gamma' \vdash t[{}^{[\alpha](v)u}/_{[\alpha]v}] : A \mid \Delta \cup \Delta', \alpha : C$.

Proposition 7 (Subject reduction)
If $\Gamma \vdash t : A \mid \Delta$ *is derivable and* $t \to t'$*, then* $\Gamma' \vdash t' : A \mid \Delta'$ *is derivable where* Γ' *(resp.* Δ'*) is the subset of* Γ *(resp.* Δ*) containing only the typing declarations for the free* λ*-variables (resp.* μ*-variables) of* t'*.*

PROOF: See [14]. □

Proposition 8 (Strong normalization)
If $\Gamma \vdash t : A \mid \Delta$ *is derivable, there is no infinite sequence of reductions starting from* t*.*

PROOF: See [15]. □

4.3 Types for the KAM

We extend all the typing judgments with a context in the right-hand side and typing judgments for environments become $\Gamma \vdash_{\texttt{env}} e : \{\Xi \mid \Theta\} \mid \Delta$ where Θ contains typing declarations for the μ-variables defined in e.

The unique new rule is the following one:

$$\frac{\Gamma \vdash_{\texttt{env}} e : \{\Xi \mid \Theta\} \mid \Delta \qquad \Gamma' \vdash_{\texttt{stack}} \pi : A^{\perp} \mid \Delta'}{\Gamma \cup \Gamma' \vdash_{\texttt{env}} e + (\alpha = \pi) : \{\Xi \mid \Theta, \alpha : A\} \mid \Delta \cup \Delta'}$$

the other ones are extended in the natural way with right-hand side contexts.

Lemma 10 (Typing and expansion)
If $\Gamma \vdash_{\texttt{state}} s \mid \Delta$ *is derivable, there exists an atom* X *such that* $\Gamma' \vdash \widetilde{s} : X \mid \Delta'$ *is derivable where* Γ' *(resp.* Δ'*) is the subset of* Γ *(resp.* Δ*) containing only the typing declarations for the free* λ*-variables (resp.* μ*-variables) of* $\widetilde{s}$*.*

PROOF: As for lemma 6, we first prove the closure case: if $\Gamma \vdash_{\texttt{clos}} (t, e) : A \mid \Delta$ is derivable, then $\Gamma' \vdash t\{e\} : A \mid \Delta'$. We use the same notations so that $\Gamma_1 \vdash t : A \mid \Delta_1$ and $\Gamma_2 \vdash_{\texttt{env}} e : \{\Xi \mid \Theta\} \mid \Delta_2$, and we just prove the case $e = e' + (\alpha = \pi)$. From $\Gamma_2 \vdash_{\texttt{env}} e : \{\Xi \mid \Theta\} \mid \Delta_2$ we can deduce $\Gamma_2 = \Gamma'_2 \cup \Gamma''_2$, $\Delta_2 = \Delta'_2 \cup \Delta''_2$ and $\Theta = \Theta', \alpha : B$ with $\Gamma'_2 \vdash_{\texttt{env}} e' : \{\Xi \mid \Theta'\} \mid \Delta'_2$ and $\Gamma''_2 \vdash_{\texttt{stack}} \pi : B^{\perp} \mid \Delta''_2$. If $\pi = c_1 :: \cdots :: c_n :: \varepsilon$ and $B = B_1 \to \cdots \to B_n \to X$, we have for each $1 \le i \le n$, $\Gamma''_{2,i} \vdash_{\texttt{clos}} c_i : B_i \mid \Delta''_{2,i}$ with $\Gamma''_2 = \Gamma''_{2,1} \cup \cdots \cup \Gamma''_{2,n}$ and by induction hypothesis $\Gamma'''_{2,i} \vdash \widetilde{c_i} : B_i \mid \Delta'''_{2,i}$. By induction hypothesis we can deduce from $\Gamma_1 \vdash t : A \mid \Delta_1$ and $\Gamma'_2 \vdash_{\texttt{env}} e' : \{\Xi \mid \Theta'\} \mid \Delta'_2$ that $\Sigma \vdash t\{e'\} : A \mid \Pi$ (for some $\Sigma \subset (\Gamma_1 \setminus \Xi) \cup \Gamma'_2$ and for some $\Pi \subset (\Delta_1 \setminus \Theta') \cup \Delta'_2$). If α is not free in t this is enough to conclude and if α is free in t, $\Pi = \Pi', \alpha : B$ so that $\Sigma \cup \Gamma'''_2 \vdash t\{e'\}[{}^{[\alpha](v)\widetilde{\pi}}/_{[\alpha]v}] : A \mid \Pi' \cup \{\alpha : X\} \cup \Delta'''_2$ by lemma 9.

If $\Gamma \vdash_{\texttt{state}} s \mid \Delta$ is derivable with $s = \langle\, t \ , \ e \ , \ \pi \,\rangle$, we easily conclude with the closure case. □

Proposition 9 (Subject reduction)
The evaluation of the KAM *preserves typing,* i.e. *if* $\Gamma \vdash_{\texttt{state}} \langle t \ , \ e \ , \ \pi \rangle \mid \Delta$ *is derivable and if the following transition is valid:*

$$\langle t \ , e \ , \pi \rangle \quad \longrightarrow \quad \langle t' \ , e' \ , \pi' \rangle$$

then $\Gamma' \vdash_{\texttt{state}} \langle t' \ , \ e' \ , \ \pi' \rangle \mid \Delta'$ *where* Γ' *(resp.* Δ'*) is the subset of* Γ *(resp.* Δ*) containing only the typing declarations for the free* λ*-variables (resp.* μ*-variables) of* $\langle t' \ , \ e' \ , \ \pi' \rangle$*.*

Proof: If $\Gamma \vdash_{\texttt{state}} \langle t \ , \ e \ , \ \pi \rangle \mid \Delta$ is derivable, we must have $\Gamma = (\Gamma_1 \setminus \Xi) \cup \Gamma_2 \cup \Gamma_3$ and $\Delta = (\Delta_1 \setminus \Theta) \cup \Delta_2 \cup \Delta_3$ with $\Gamma_1 \vdash t : A \mid \Delta_1$, $\Gamma_2 \vdash_{\texttt{env}} e : \{\Xi \mid \Theta\} \mid \Delta_2$ and $\Gamma_3 \vdash_{\texttt{stack}} \pi : A^{\perp} \mid \Delta_3$. The proof for the transitions (*push*), (*pop*) and (*deref*) is almost the same as for proposition 4, and we just give the two other cases:

(*save*) We have $t = \mu\alpha.t'$, this entails $\Delta_1 = \Delta'_1 \setminus \{\alpha : A\}$ with $\Gamma_1 \vdash t' : \bot \mid \Delta'_1$, and we can derive:

$$\dfrac{\dfrac{\Gamma_1 \vdash t' : \bot \mid \Delta'_1 \qquad \dfrac{\Gamma_2 \vdash_{\texttt{env}} e : \{\Xi \mid \Theta\} \mid \Delta_2 \qquad \Gamma_3 \vdash_{\texttt{stack}} \pi : A^{\perp} \mid \Delta_3}{\Gamma_2 \cup \Gamma_3 \vdash_{\texttt{env}} e + (\alpha = \pi) : \{\Xi \mid \Theta, \alpha : A\} \mid \Delta_2 \cup \Delta_3}}{(\Gamma_1 \setminus \Xi) \cup \Gamma_2 \cup \Gamma_3 \vdash_{\texttt{clos}} (t', e + (\alpha = \pi)) : \bot \mid (\Delta'_1 \setminus (\Theta \cup \{\alpha : A\})) \cup \Delta_2 \cup \Delta_3} \qquad \dfrac{}{\vdash_{\texttt{stack}} \varepsilon : \top \mid}}{(\Gamma_1 \setminus \Xi) \cup \Gamma_2 \cup \Gamma_3 \vdash_{\texttt{state}} \langle t' \ , \ e + (\alpha = \pi) \ , \ \varepsilon \rangle \mid (\Delta'_1 \setminus (\Theta \cup \{\alpha : A\})) \cup \Delta_2 \cup \Delta_3}$$

with $\Delta'_1 \setminus (\Theta \cup \{\alpha : A\}) = (\Delta'_1 \setminus \{\alpha : A\}) \setminus \Theta = \Delta_1 \setminus \Theta$ and s and s' have the same free variables.

(*restore*) We have $t = [\alpha]t'$ and $\pi = \varepsilon$, this entails $\Delta_1 = \Delta'_1, \alpha : B$ and $A = \bot$, and Γ_3 and Δ_3 are empty with $\Gamma_1 \vdash t' : B \mid \Delta'_1$. Moreover $e = e_0 + (\alpha = \pi')$ so that $\Gamma_2 = \Gamma'_2 \cup \Gamma''_2$, $\Delta_2 = \Delta'_2 \cup \Delta''_2$ and $\Theta = \Theta', \alpha : B$ with $\Gamma'_2 \vdash_{\texttt{env}} e_0 : \{\Xi \mid \Theta'\} \mid \Delta'_2$ and $\Gamma''_2 \vdash_{\texttt{stack}} \pi' : B^{\perp} \mid \Delta''_2$, and we can derive:

$$\dfrac{\dfrac{\Gamma_1 \vdash t' : B \mid \Delta'_1 \qquad \Gamma_2 \vdash_{\texttt{env}} e : \{\Xi \mid \Theta\} \mid \Delta_2}{(\Gamma_1 \setminus \Xi) \cup \Gamma_2 \vdash_{\texttt{clos}} (t', e) : B \mid (\Delta'_1 \setminus \Theta) \cup \Delta_2} \qquad \Gamma''_2 \vdash_{\texttt{stack}} \pi' : B^{\perp} \mid \Delta''_2}{(\Gamma_1 \setminus \Xi) \cup \Gamma_2 \vdash_{\texttt{state}} \langle t' \ , \ e \ , \ \pi' \rangle \mid (\Delta'_1 \setminus \Theta) \cup \Delta_2}$$

and s and s' have the same free λ-variables and their free μ-variables can only differ on α if $\alpha \notin t'$. □

Corollary 9.1 (Termination)
If $\Gamma \vdash_{\texttt{state}} s \mid \Delta$ *is derivable, the* KAM*, starting from the state* s*, stops in a state* $\langle x \ , \ e' \ , \ \pi' \rangle$ *where* x *is the head* λ*-variable of the weak head normal form of* $\widetilde{s}$ *or in a state* $\langle [\alpha]u \ , \ e' \ , \ \varepsilon \rangle$ *where* α *is the head* μ*-variable of the weak head normal form of* $\widetilde{s}$*.*

Proof: We first show that the machine stops. The size $|e|$ of an environment e is now the sum of the sizes of its closures plus the sum of the sizes of its stacks, and the size $|s|$ of a stack s is the sum of the sizes of its closures (with $|\varepsilon| = 0$). The transitions which don't correspond to a reduction step of the term associated to the state are: (*push*), (*deref*), (*restore*) and also (*save*) when the stack is empty, but we can't have an infinite sequence of such transitions since they decrease the value of $(|e|, |t|)$. We can deduce that an infinite sequence of transitions of the machine contains an infinite number of transitions (*pop*) and (non-degenerated) (*save*) so that, with proposition 6, the term associated with the starting state would have an infinite sequence of reductions contradicting lemma 10 and proposition 8.

Let s' be the stopping state, it can be:

- $\langle\, \lambda x.u \,,\, e' \,,\, \varepsilon \,\rangle$, impossible by proposition 9 since it must be typable and ε is not typable of a type $A \wedge P$.
- $\langle\, x \,,\, e' \,,\, \pi' \,\rangle$ with x not declared in e' and $(x\{e'\})\widetilde{\pi'} = (x)\widetilde{\pi'}$, so that, by corollary 6.1, x is the head λ-variable of the weak head normal form of $\widetilde{s}$.
- $\langle\, [\alpha]u \,,\, e' \,,\, \pi' \,\rangle$ with $\pi' \neq \varepsilon$, impossible by proposition 9 since it must be typable and $\pi' \neq \varepsilon$ is not typable of type $\top$.
- $\langle\, [\alpha]u \,,\, e' \,,\, \varepsilon \,\rangle$ with α not declared in e' and α is the head μ-variable of $[\alpha](u\{e'\})$. $\square$

In particular, the machine never stops because the $\lambda\mu$-term is $[\alpha]t$ and the stack is not empty, the $\perp$ type ensures that in this case the stack is empty.

A more technical presentation of Krivine's abstract machine as an evaluation machine for the $\lambda\mu$-calculus has been given by P. de Groote [5] and a similar work has been carried out in the call-by-value case by G. Bierman [2]. A reconstruction of Krivine's abstract machine for the $\lambda\mu$-calculus through its continuation semantics has been described by T. Streicher and B. Reus [19], going in the opposite direction of our presentation.

Acknowledgments

I would like to thank P. Baillot, C. Berline, C. Fouqueré and J. Vauzeilles who read earlier versions of this work and of course the anonymous referees. The comments of all of them helped me to make this paper more readable, I hope it is now accessible to a wider audience.

References

[1] Henk Barendregt. *The lambda calculus, its syntax and semantics.* Number 103 in Studies in Logic and the Foundations of Mathematics. North-Holland, second edition, 1984.

[2] 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.

[3] Pierre-Louis Curien and Hugo Herbelin. The duality of computation. In *Proceedings of the International Conference on Functional Programming*, volume 35(9) of *ACM SIGPLAN Notices*, pages 233–243. Association for Computing Machinery, ACM Press, September 2000.

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

[5] Philippe de Groote. An environment machine for the lambda-mu-calculus. *Mathematical Structures in Computer Science*, 8:637–669, 1998.

[6] Matthias Felleisen and Robert Hieb. The revised report on the syntactic theories of sequential control and state. *Theoretical Computer Science*, 103(2):235–271, September 1992.

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

[8] Timothy Griffin. A formulae-as-types notion of control. In *Proceedings of the 1990 Principles of Programming Languages Conference*, pages 47–58. IEEE Computer Society Press, 1990.

[9] Martin Hofmann and Thomas Streicher. Completeness of continuation models for lambda-mu-calculus. *Information and Computation*, 179(2):332–355, December 2002.

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

[11] Jean-Louis Krivine. *Lambda-calculus, types and models.* Ellis Horwood, 1993.

[12] Jean-Louis Krivine. Typed lambda-calculus in classical Zermelo-Fraenkel set theory. *Archive for Mathematical Logic*, 40(3):189–205, 2001.

[13] Mark Lillibridge. Unchecked exceptions can be strictly more powerful than call/cc. *Higher-Order and Symbolic Computation*, 12(1):75–104, April 1999.

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

[15] Michel Parigot. Strong normalization for second order classical natural deduction. *Journal of Symbolic Logic*, 62(4):1461–1479, December 1997.

[16] Jon Riecke and Hayo Thielecke. Typed exceptions and continuations cannot macro-express each other. In J. Wiedermann, P. van Emde Boas, and M. Nielsen, editors, *International Colloquium on Automata, Languages and Programming*, volume 1644 of *Lecture Notes in Computer Science*, pages 635–644. Springer, July 1999.

[17] Paul Rozière. Déduction naturelle classique et $\lambda\mu$-calcul. Notes de cours de DEA. Available at `http://www.pps.jussieu.fr/~roziere/dea/classicalded.pdf` , 2001.

[18] Peter Selinger. Control categories and duality: on the categorical semantics of the lambda-mu calculus. *Mathematical Structures in Computer Science*, 11(2):207–260, April 2001.

[19] Thomas Streicher and Bernhard Reus. Classical logic, continuation semantics and abstract machines. *Journal of Functional Programming*, 8(6):543–572, November 1998.

[20] Hayo Thielecke. Comparing control constructs by double-barrelled cps. *Higher-Order and Symbolic Computation*, 15(2/3):141–160, September 2002.