A machine for programs extracted with the axiom of choice

Jean-Louis Krivine

PPS Group, University Paris 7, CNRS

krivine@pps.jussieu.fr

Birmingham, May 17, 2005

Introduction

My motive for considering a λ-calculus head reduction machine :
Extend the Curry-Howard (proof-program) correspondence
to the whole of mathematics.

1st problem

Each mathematical *proof* must give a *program*
which must be executable in such a machine.

2nd problem

To understand the behaviour of these programs
i.e. the *specification* associated with a given *theorem*.

The first problem is now completely solved, but the second is far from being so.

A machine in symbolic form

The machine is the program side of the proof-program correspondence.
In this talk, I use only a machine in symbolic form,
not an explicit implementation.
We execute a *process* $t \star \pi$; t is (provisionally) a closed λ-term,
π is a *stack*, that is a sequence $t_1.t_2 \dots t_n.\pi_0$ where
π_0 is a *stack constant*, i.e. a marker for the bottom of the stack.

Execution rules for processes :

$tu \star \pi \succ t \star u.\pi$ (*push*)

$\lambda x\, t \star u.\pi \succ t[u/x] \star \pi$ (*pop*)

This symbolic machine will be used to follow the execution of programs
written in an extension of λ-calculus with new instructions.

A machine in symbolic form (cont.)

We get a better approximation of a "real" machine by eliminating substitution.

The execution rules are a little more complicated :

$tu \star \pi \succ t \star u.\pi$

$\lambda x_1 \ldots \lambda x_k t u \star t_1 \ldots t_k.\pi \succ \lambda x_1 \ldots \lambda x_k t \star t_1 \ldots t_k.v.\pi$

with $v = (\lambda x_1 \ldots \lambda x_k u) t_1 \ldots t_k$

$\lambda x_1 \ldots \lambda x_k x_i \star t_1 \ldots t_k.\pi \succ t_i \star \pi.$

It is necessary to add new instructions, because such simple machines can only handle ordinary λ-terms, i.e. programs obtained from proofs in *pure intuitionistic logic*.

Observe that some of these instructions will be *incompatible with β-reduction.*

Realizability

We know that proofs in pure intuitionistic logic give λ-terms.
But *pure intuitionistic*, or even *classical*, logic is not sufficient
to write down mathematical proofs.
We need *axioms*, such as *extensionality, infinity, choice, ...*
Axioms are not theorems, they have no proof !
How can we find suitable programs for them ?

The solution is given by the theory of classical realizability
by means of which we define, for each mathematical formula Φ :

- the set of stacks which *are against* Φ, denoted by $\|\Phi\|$
- the set of closed terms t which *realize* Φ, which is written $t \Vdash \Phi$.

We first choose a set of processes, denoted by $\perp\!\!\!\perp$, which is *saturated*, i.e.
$t \star \pi \in \perp\!\!\!\perp,\ t' \star \pi' \succ t \star \pi \ \Rightarrow\ t' \star \pi' \in \perp\!\!\!\perp.$

Realizability (cont.)

The set $\|\Phi\|$ and the property $t \Vdash \Phi$ are defined by induction on the formula Φ.
They are connected as follows :

$$t \Vdash \Phi \Leftrightarrow (\forall \pi \in \|\Phi\|)\, t \star \pi \in \perp\!\!\!\perp$$

Two steps of induction, because we use only two logical symbols : $\to$, $\forall$.

1. $\|\Phi \to \Psi\| = \{t.\pi \,;\, t \Vdash \Phi, \pi \in \|\Psi\|\}$. In words :
if the term t realizes the formula Φ and the stack π is against the formula Ψ
then the stack $t.\pi$ (push t on the top of π) is against the formula $\Phi \to \Psi$.

2. $\|\forall x\, \Phi(x)\| = \bigcup_{a \in A} \|\Phi(a)\|$ where A is the domain of the variable x
(it may be the integers, or the whole universe of sets, ...).
In words : a stack is against $\forall x\, \Phi(x)$ if it is against $\Phi(a)$ for some a.
It follows that $t \Vdash \forall x\, \Phi(x) \Leftrightarrow t \Vdash \Phi(a)$ for all a.

The language of mathematics

The proof-program correspondence is well known for *intuitionistic logic*. Now we have

Mathematics $\equiv$ Classical logic + some axioms that is

Mathematics $\equiv$ Intuitionistic logic + Peirce's law + some axioms

For each axiom $\mathcal{A}$, we choose a closed λ-term which realizes $\mathcal{A}$, *if there is one*.

If not, *we extend our machine* with some new instruction which realizes $\mathcal{A}$,

if we can devise such an instruction.

Now, there are essentially two possible axiom systems for mathematics :

1. *Analysis*, i.e. second order classical logic with dependent choice.
2. *ZFC*, i.e. Zermelo-Fraenkel set theory with the full axiom of choice.

Let us look more closely at these axioms and the instructions for them.

In this talk, I only give the results, without proof.

Axioms for mathematics

The law of Peirce is $(\neg A \to A) \to A$.

• Analysis is composed of three groups of axioms :

i) Equations such as $x + 0 = x$, $x + sy = s(x + y)$, ...

and inequations such as $s0 \neq 0$.

ii) The recurrence axiom, which says that each individual (1st order object) is an integer.

In fact (and fortunately) we need only a strictly weaker axiom :

Every non void set of individuals has an element without predecessor in this set.

iii) The axiom of dependent choice :

If $\forall X \exists Y\ F(X, Y)$, then there exists a sequence X_n such that $F(X_n, X_{n+1})$.

• ZFC has two groups of axioms :

The axioms of ZF, which I don't want to list.

The full axiom of choice : Any product of non void sets is non void.

Peirce's law

We adapt to our machine the solution found by Tim Griffin in 1990.

We add to the λ-calculus an instruction denoted by cc. Its reduction rule is :

$$\mathsf{cc} \star t.\pi \succ t \star \mathsf{k}_\pi.\pi$$

k_π is a *continuation*, that is to say a pointer to a location where the stack is saved.

In our symbolic machine, it is simply a λ-constant, indexed by π.

Its execution rule is $\mathsf{k}_\pi \star t.\pi' \succ t \star \pi$.

Therefore cc saves the current stack and k_π restores it.

Using the theory of classical realizability, we can show that

$$\mathsf{cc} \Vdash ((\neg A \to A) \to A).$$

In this way, we have extended the Curry-Howard correspondence

to every proof in *pure* (i.e. without axiom) *classical logic*.

Example : proof of $\exists x(Px \to \forall y\, Py)$

Write this theorem $\forall x[(Px \to \forall y\, Py) \to X] \to X$ (by definition of $\exists$).
We must show $z : \forall x[(Px \to \forall y\, Py) \to X] \vdash ? : X$
The hypothesis $k : \neg X$ gives $k \circ z : (Px \to \forall y\, Py) \to Px$, then $\mathsf{cc}\, k \circ z : Px$
We get $\mathsf{cc}\, k \circ z : \forall x\, Px$, then $\lambda d\, \mathsf{cc}\, k \circ z : Px \to \forall y\, Py$
then $k : \neg X \vdash z \lambda d\, \mathsf{cc}\, k \circ z : X$ and finally $\mathsf{cc}\, \lambda k\, z \lambda d\, \mathsf{cc}\, k \circ z : X$.
We have obtained the program $\theta = \lambda z\, \mathsf{cc}\, \lambda k\, z \lambda d\, \mathsf{cc}\, \lambda x(k)(z)x$.
Look at its behavior, in a process $\theta \star t.\pi$. We have $\theta \star t.\pi \succ t \star \alpha_{t,\pi}.\pi$
with $\alpha_{t,\pi} = \lambda d\, \mathsf{cc}\, \lambda x(\mathsf{k}_\pi)(t)x$. So, $\alpha_{t,\pi}$ is a dynamic instruction
with the reduction rule : $\alpha_{t,\pi} \star u.\rho \succ t \star \mathsf{k}_\rho.\pi$.
Let α be a constant and assume that $t \star \alpha.\pi \succ \alpha \star u_0[\alpha].\rho_0[\alpha]$.
Thus $t \star \mathsf{k}_\rho.\pi \succ \mathsf{k}_\rho \star u_0[\mathsf{k}_\rho].\rho_0[\mathsf{k}_\rho] \succ u_0[\mathsf{k}_\rho] \star \rho$.
We have, in fact, the following reduction rule : $\alpha_{t,\pi} \star u.\rho \succ u_0[\mathsf{k}_\rho] \star \rho$
(note that u_0 is computed when α comes in head position for the first time).

The recurrence axiom

Equations like $x + 0 = x$ and inequations like $0 \neq 1$ are very easy to realize.

Indeed $\lambda x\, x \Vdash x + 0 = x$ and $\lambda x\, xt \Vdash 0 \neq 1$ (t is arbitrary).

But the proper recurrence axiom, that is the following formula :

$\forall X[X0, \forall x(Xx \to Xsx) \to \forall x\, Xx]$ is *impossible to realize*.

Fortunately, we really need a weaker formula : strengthen the hypothesis $X0$,

by saying that the set X contains *every individual which is not a successor*.

We can write this axiom in a better form :

$$\forall x[\forall y(Xy \to x \neq sy) \to \neg Xx] \to \forall x\, \neg Xx$$

It is realized by any fixed point combinator Y, with the following reduction rule :

$$\mathsf{Y} \star t.\pi \succ t \star \mathsf{Y}t.\pi.$$

We can take $\mathsf{Y} = \mathsf{ZZ}$ with $\mathsf{Z} = \lambda z \lambda x(x)(z)zx$ (Turing fixpoint).

The axiom of dependent choice

We need a *new instruction* in our machine. Any of the following two will work :

1. The signature. Let $t \mapsto n_t$ be a function from closed terms into the integers, which is *very easily computable* and *"practically" one-to-one*. It means that the one-to-one property has to be true only for the terms which appear during the execution of a given process. And also that we never try to compute the inverse function.
We define an instruction σ with the following reduction rule :

$$\sigma \star t.\pi \succ t \star \underline{n_t}.\pi.$$

A simple way to implement such an instruction is to take for n_t the *signature* of the term t, given by a standard algorithm, such as MD5 or SHA1.
Indeed, these functions are almost surely one-to-one for the terms which appear during a finite execution of a given process.

The axiom of dependent choice (cont.)

2. The clock. It is denoted as $\hbar$ and its reduction rule is :

$$\hbar \star t.\pi \succ t \star \underline{n}.\pi$$

where $\underline{n}$ is a Church integer which is the current time (for instance, the number of reduction steps from the boot).

Both instructions, the clock and the signature, can be given (realize) the same type, which is not DC but a formula DC' which *implies DC in classical logic*.
By means of this proof, we get a λ-term $\gamma[\mathsf{cc}, \sigma]$ or $\gamma[\mathsf{cc}, \hbar]$ which has the type DC.
The instructions σ, $\hbar$ appear only inside this λ-term γ.
By looking at its behavior, we find that the integers produced by these instructions are only compared with each other. No other operation is performed on these integers.

A program for the axiom DC

The explicit writing of the program $\gamma[\mathsf{cc}, \sigma]$ of type DC is as follows :

$$\gamma = \lambda f(\sigma)(\mathsf{Y})\lambda x\lambda n(\mathsf{cc})\lambda k\, f\tau_0\tau_1$$

with $\tau_0 = \lambda v\, vxnk$, $\tau_1 = \lambda u\lambda x'\lambda n'\lambda k'\, \mathsf{Comp}\, nn'\alpha\alpha' u$,

$\alpha = (k)(x')n$, $\alpha' = (k')(x)n'$,

$\mathsf{Comp}\, nn'\alpha\alpha' u = \alpha$ if $n < n'$, α' if $n' < n$, u if $n = n'$.

Consider a process $\gamma \star f.\pi$ in which γ is in head position. We have :

$\gamma \star f.\pi \succ \sigma \star \mathsf{Y}\xi_f.\pi$ where $\xi_f = \lambda x\lambda n(\mathsf{cc})\lambda k\, f\tau_0\tau_1$ depends only on f

$\succ \mathsf{Y}\xi_f \star n_f.\pi \succ \xi_f \star \eta_f.n_f.\pi$, with $\eta_f = \mathsf{Y}\xi_f$. Therefore

$$\gamma \star f.\pi \succ f \star \tau_0^{f\pi}.\tau_1^{f\pi}.\pi$$

with $\tau_i^{f\pi} = \tau_i[\eta_f/x, n_f/n, k_\pi/k]$.

A program for the axiom DC (cont.)

Now $\tau_0^{f\pi}$ is simply the triple $<\eta_f, n_f, k_\pi>$. In other words
$\tau_0^{f\pi}$ stores the current state $f.\pi$ when γ comes in head position.
$\tau_1^{f\pi}$ performs the real job : it looks at two such states $f.\pi$ and $f'.\pi'$ and compare the indexes n_f and $n_{f'}$. If $n_f = n_{f'}$ it does nothing.
If $n_f < n_{f'}$ (resp. $n_{f'} < n_f$) it restarts with $\gamma \star f'.\pi$ (resp. $\gamma \star f.\pi'$) :
in each case, *the second file with the first stack*.
Thus, the main function of this program is to *update files* (if σ is a clock)
or *to choose a good version of a file* (if σ is a signature).

The axiom of dependent choice is a very general and useful principle in mathematics. Its informatic translation is also a very general program to update files or choose the suitable release of a file.

Zermelo-Fraenkel set theory

Axioms of ZFC can be classified in three groups :

1. Equality, extensionality, foundation.
2. Union, power set, substitution, infinity.
3. Choice.

I will not give details about λ-terms which realize the first two groups.
Observe simply that *they are λ-terms*, i.e. no new instruction is necessary.
Curiously, equality and extensionality are the most difficult ones. For example,
the first axiom of equality $\forall x(x = x)$ is realized by a λ-term τ
with the reduction rule : $\tau \star t.\pi \succ t \star \tau.\tau.\pi$
(fixed point of $\lambda x \lambda f\ fxx$).

The full axiom of choice

The problem for the *full axiom of choice* has been solved very recently (not yet published). As a bonus, we get also the *continuum hypothesis*.
The situation is completely different for these axioms :
we need two new instructions χ and χ' which appear inside two very complex λ-terms, together with cc and the clock (or the signature).
The behaviour of these programs is, for the moment, not understood.
These instructions χ, χ' work on the *bottom of the stack*.
Their reduction rules is as follows :

$$\chi \star t.\tau.t_1 \dots t_n.\pi_0 \succ t \star t_1 \dots t_n.\tau.\pi_0$$
$$\chi' \star t.t_1 \dots t_n.\tau.\pi_0 \succ t \star \tau.t_1 \dots t_n.\pi_0$$

where π_0, as before, is a marker for the bottom of the stack.

Conclusion

The conclusion is that we can translate *every mathematical proof*
into a program. We can execute this program in a lazy λ-calculus machine
extended with only four new instructions : cc, σ (or $\hbar$), χ and χ'
which are rather easy to implement.

The challenge, now, is to understand all these programs.

References

1. **S. Berardi, M. Bezem, T. Coquand** *On the computational content of the axiom of choice.* J. Symb. Log. 63, pp. 600-622 (1998).
2. **U. Berger, P. Oliva** *Modified bar recursion and classical dependent choice.* Preprint.
3. **J.-L. Krivine** *Typed lambda-calculus in classical Zermelo-Fraenkel set theory.*
Arch. Math. Log. 40, 3, pp. 189-205 (2001)
4. **J.-L. Krivine** *Dependent choices, 'quote' and the clock.*
Th. Comp. Sc. 308, pp. 259-276 (2003)
5. **J.-L. Krivine** *Realizability in classical logic.*
To appear in Panoramas et Synthèses. Société mathématique de France.

Pdf files at http://www.pps.jussieu.fr/~krivine