Machine Deduction

Christophe Raffalli *
L.F.C.S. Edinburgh

August 1993

Abstract

We present in this paper a new type system which allows to extract code for an abstract machine instead of lambda-terms. Thus, we get a framework to compile correctly programs extracted from proof by translating their proof in our system and then extracting the code. Moreover, we will see that we can associate programs to classical proofs.

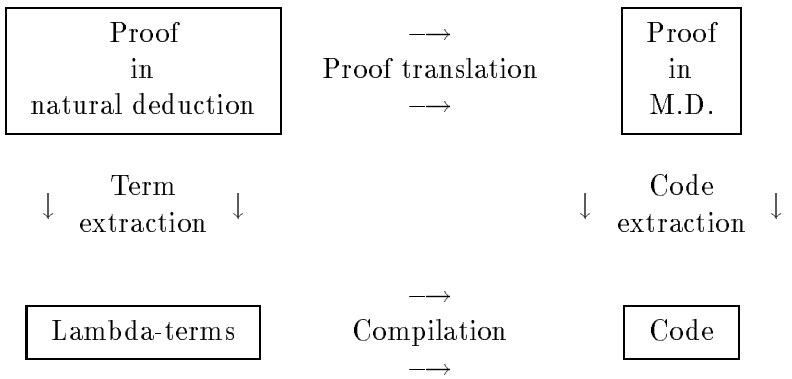
1 Introduction.

The proof as program paradigm, using the Curry-Howard isomorphism [4], gives a way to associate a program to an intuitionistic proof. This program is almost always a functional program (in general a lambda-term [1]) which has to be compiled before being executed [11]. This ensures some correctness about the functional program extracted from the proof. But the correctness of the compiled code is relative to the proof of the compiler.

The usual way to ensure this kind of correctness is to define a semantics for the functional language, and to verify that the compiler preserves this semantics.

We study in this paper a type system for the code of an abstract machine (S.E.C. machine). This approach authorizes a new kind of compilation: we translate the proof in natural deduction to a proof in our system and we extract the code from this new proof.

The two kinds of compilation can be represented by the following diagram:



To achieve this goal we define a deduction system MD_{SEC}, for intuitionistic logic. This system is a second order system specially tailored to a translation of Leivant and Krivine's system AF_2[6, 7, 9, 8].

*cr@dcs.ed.ac.uk

Secondly, we define an S.E.C. machine and an interpretation for intuitionistic logic in terms of the machine. Then, we show how to extract code from a proof in our system and we prove this code correct for our interpretation: the extracted code belongs to the interpretation of its type.

Next, we show how to deal with data types and optimization, and how the interpretation notion ensures the correctness of the code.

Finally we give one proof translation for call-by-name (in Krivine's style). This means that we can find a proof translation such that the previous diagram commutes for call-by-name compilation.

If the reader is not familiar with second order stuff and system AF_2, he could read this paper forgetting all about first order to use only its propositional part (This restriction of system AF_2 gives the Curry's system F [3]). But doing this, the reader will lose the argument about the correctness of programs in Section 6, because the type characterizes the function in system AF_2 but not in system F.

2 The deduction system.

We use classical second order formulas. First we define first order terms from a language $\mathcal{L}$ defined by an algebraic signature Σ. We choose an infinite set of predicate variables (infinite for each arity) and we construct formulas from atomic formulas, false, true, negation, conjunction, second order and first order existential quantification ($\bot \mid \top \mid X(t_1,\ldots,t_n) \mid F \wedge G \mid \neg F \mid \exists X F \mid \exists x F$). Let $\mathcal{F}$ be this set of formulas.

Definition 2.1 *We choose a partition of second order variables into stack variables ($\mathcal{V}^\Pi$) and value variables ($\mathcal{V}^\Phi$). We define value formulas ($\mathcal{F}^\Phi$) and stack formulas ($\mathcal{F}^\Pi$) as the least subsets of $\mathcal{F}$ verifying:*

$$\begin{array}{rcll} X(t_1,\ldots,t_n) & \in & \mathcal{F}^\Pi & \text{if } X \in \mathcal{V}^\Pi \\ A(t_1,\ldots,t_n) & \in & \mathcal{F}^\Phi & \text{if } A \in \mathcal{V}^\Phi \\ F \wedge P & \in & \mathcal{F}^\Pi & \text{if } F \in \mathcal{F}^\Phi \text{ and } P \in \mathcal{F}^\Pi \\ \exists \chi P & \in & \mathcal{F}^\Pi & \text{if } P \in \mathcal{F}^\Pi \text{ and } \chi \text{ is any kind of variable} \\ \neg P & \in & \mathcal{F}^\Phi & \text{if } P \in \mathcal{F}^\Pi \end{array}$$

In all this paper, we will use the following notation to write formulas:

- M, N for any formulas.
- F, G for value formulas.
- P, Q, R for stack formulas.
- X, Y for stack variables.
- A, B for value variables.
- x, y for first order variables.
- t, u for first order terms.

- Γ, Δ for multiset of value formulas.
- $M[t/x]$ for the substitution of the first order variable x with a term t in the formula M.
- $M[\lambda x_1 \ldots \lambda x_n N/X]$ for the substitution of the the second order variable X of arity n with a formula N in the formula M. This substitution is defined as usual, but to be compatible with the definition of stack formulas and value formulas, we will substitute only stack formulas to stack variables and value formulas to value variables.
- χ for any kind of variable (first order, stack or value variables).
- φ, associated to χ, for an expression which is substitutable to χ:
  - φ is a term if χ is a first order variable
  - $\varphi = \lambda x_1 \ldots \lambda x_n P$ where P is a stack formula if χ is a stack variable or arity n
  - $\varphi = \lambda x_1 \ldots \lambda x_n F$ where F is a value formula if χ is a value variable or arity n

We will consider only sequents of the following form, with $F_1, \ldots, F_n \in \mathcal{F}^{\Phi}$ and $P \in \mathcal{F}^{\Pi}$:

$$F_1, \ldots, F_n \mid P \vdash$$

In such a sequent, the "," and "|" must be understand as conjunction. So $F_1, \ldots, F_n \mid P \vdash$ means that we get a contradiction from $F_1 \wedge \ldots \wedge F_n \wedge P$.

Here are the rules of the deduction system MD_{SEC}:

$$\frac{}{F_1, \ldots, F_{m-1}, \neg P, F_{m+1}, \ldots, F_n \mid P \vdash} Ax^{\Gamma} \qquad \frac{F_1, \ldots, F_m, \ldots, F_n \mid (F_m \wedge P) \vdash}{F_1, \ldots, F_m, \ldots, F_n \mid P \vdash} Co$$

$$\frac{F_1, \ldots, F_n \mid P \vdash}{\Gamma \mid F_1 \wedge \ldots \wedge F_n \wedge P \vdash} \wedge_i \qquad \frac{\Gamma \mid \neg Q \wedge P \vdash \quad \Gamma \mid Q \vdash}{\Gamma \mid P \vdash} \wedge\neg_e$$

$$\frac{\Gamma \mid \top \vdash}{\Gamma \mid P \vdash} \top_e \qquad \frac{\Gamma \mid \neg(\neg P \wedge \top) \wedge P \vdash}{\Gamma \mid P \vdash} \top'_e$$

$$\frac{\Gamma \mid P \vdash \qquad \chi \notin \Gamma}{\Gamma \mid \exists \chi\, P \vdash} \exists_i \qquad \frac{\Gamma \mid \exists \chi\, P \vdash}{\Gamma \mid P[\varphi/\chi] \vdash} \exists_e$$

$$\frac{}{\Gamma \mid \bot \vdash} \bot_s$$

Note: the $\top$ connective is not useful when we use existential quantifier. If we replace $\top$ by $\exists X\, X$ the rule $\top_e$ is derivable and the rule $\bot_s$ is still correct. We give a system using $\top$ to have also a complete propositional version.

Proposition 2.2 *We remark that this deduction system is correct for intuitionistic logic. This means that if we prove $\Gamma \mid P \vdash$ in our system then $\Gamma, P \vdash$ is a valid sequent in intuitionistic logic.*

proof: The proof is done by induction on the proof of $\Gamma \mid P \vdash$. In fact, it's sufficient to remark that all rules are correct (the rule $\top'_e$ could seem classical. But because formulas are to the left, this rule is in fact equivalent to $\neg(\neg\neg P \wedge P) \rightarrow \neg P$ which is intuitionisticaly true). ♠

3 Programs and machine.

In this section, we define an S.E.C. machine. We define the set of instructions $\mathcal{I}$, the set of programs $\mathcal{P}$, the set of environments $\mathcal{E}$, the set of values $\mathcal{V}$ and the set of stacks $\mathcal{S}$ as the least sets verifying the following conditions:

$$
\begin{array}{rcll}
\texttt{Jump}_n & \in & \mathcal{I} & \text{if } n \in \mathbb{N} \\
\texttt{Pop}_n & \in & \mathcal{I} & \text{if } n \in \mathbb{N} \\
\texttt{Push}[p] & \in & \mathcal{I} & \text{if } p \in \mathcal{P} \\
\texttt{Push}_n & \in & \mathcal{I} & \text{if } n \in \mathbb{N} \\
\texttt{Erase} & \in & \mathcal{I} & \\
\texttt{Stop} & \in & \mathcal{I} & \\
\texttt{Save} & \in & \mathcal{I} & \\
\texttt{Rest} & \in & \mathcal{I} & \\
\mathcal{P} & = & \mathcal{I}^{(\mathbb{N})} & (\mathcal{P} \text{ is the set of finite sequences of instructions}) \\
\mathcal{E} & = & \mathcal{V}^{(\mathbb{N})} & (\mathcal{E} \text{ is the set of finite sequences of values}) \\
\mathcal{V} & = & \mathcal{P} \times \mathcal{E} & \\
\mathcal{S} & = & \mathcal{V}^{(\mathbb{N})} & (\mathcal{S} \text{ is the set of finite sequences of values})
\end{array}
$$

We will use the following notation:

- $i;p$ for the concatenation of the instruction i at the beginning of the program p. we won't use the empty program and we will denote i the program using only one instruction i.
- $(\varphi_1, \varphi_2, \ldots, \varphi_{n-1}, \varphi_n)$ for an environment of length n.
- $()$ for an empty environment.
- $\langle p/e \rangle$ for a value with the program p and the environment e.
- $\varphi \cdot \pi$ for the concatenation of the value φ at the beginning of the stack π.
- ε for the empty stack.

Now, we define the transition function "tr" (this is a partial function), from $\mathcal{P} \times \mathcal{E} \times \mathcal{S}$ to itself. We give this definition by the table 1.

For instructions `Save` and `Rest`, we use the canonical isomorphism between $\mathcal{S}$ and $\mathcal{E}$ to store a stack in place of an environment.

Definition 3.1 *We define the partial function* $\mathrm{ex}(\varphi, \pi)$ *from* $\mathcal{V} \times \mathcal{S}$ *to* $\mathcal{S}$. *Given a value* $\varphi = \langle p/e \rangle$ *and a stack* π, *let be* $\{S_n\}_{n \in \mathbb{N}}$ *the sequence defined by*

- $S_0 = (p, e, \pi)$
- $S_{n+1} = \mathrm{tr}(S_n)$

Then,$\mathrm{ex}(\varphi, \pi)$ *is defined if the previous sequence is well defined and if exists an integer* N *such that* $S_N = (\texttt{Stop};p', e', \pi')$. *In this case, we define* $\mathrm{ex}(\varphi, \pi) = \pi'$.

Proposition 3.2 *If* $\mathrm{tr}(p, e, \pi) = (p', e', \pi')$ *then* $\mathrm{ex}(\langle p/e \rangle, \pi)) = \mathrm{ex}(\langle p'/e' \rangle, \pi')$.

proof: the proof comes from the definitions of tr and ex. ♠

Table 1: Here is the complete transition table for the S.E.C. machine:

input code output code	input environment output environment	input stack output stack
$\mathtt{Pop}_m;p$ p	e $(\varphi_1,\ldots,\varphi_m)$	$\varphi_1\cdots\varphi_m\cdot\pi$ π
$\mathtt{Push}[p'];p$ p	e e	π $\langle p'/e\rangle\cdot\pi$
$\mathtt{Push}_m;p$ p	$(\varphi_1,\ldots,\varphi_n)$ $(\varphi_1,\ldots,\varphi_n)$	π $\varphi_m\cdot\pi$
$\mathtt{Jump}_m;p$ p'	$(\varphi_1,\ldots,\varphi_{m-1},\langle p'/e'\rangle,\varphi_{m+1},\ldots,\varphi_n)$ e'	π π
$\mathtt{Erase};p$ p	e e	π ε
$\mathtt{Stop};p$ $\mathtt{Stop};p$	e e	π π
$\mathtt{Save};p$ p	e e	π $\langle\mathtt{Rest}/\pi\rangle\cdot\pi$
$\mathtt{Rest};p$ p'	$e=\pi'$ e'	$\langle p'/e'\rangle\cdot\pi$ π'

4 Semantics.

Definition 4.1 *An interpretation σ, is given by:*

- *A mapping $x\mapsto|x|^\sigma$ from first order variables to $\mathcal{V}$.*
- *A mapping $x\mapsto|f|^\sigma$ from function constants of arity n to $\mathcal{V}^n\to\mathcal{V}$.*
- *A mapping $A\mapsto|A|^\sigma$ from program variables of arity n to $\mathcal{V}^n\to\mathcal{P}(\mathcal{V})$.*
- *A mapping $X\mapsto|X|^\sigma$ from stack variables of arity n to $\mathcal{V}^n\to\mathcal{P}(\mathcal{S})$.*
- *An element $|\bot|^\sigma$ of $\mathcal{S}^\sigma$.*

Definition 4.2 *Given $\Phi\in\mathcal{P}(\mathcal{V})$ and $\Pi\in\mathcal{P}(\mathcal{S})$, we define*

$$\Phi\times\Pi=\{\varphi\cdot\pi;\varphi\in\Phi\quad\textit{and}\quad\pi\in\Pi\}$$

Definition 4.3 *Given Π in $\mathcal{P}(\mathcal{S})$, we define the set $\overline{\Pi}$ in $\mathcal{P}(\mathcal{V})$ by:*

$$\overline{\Pi}=\{\varphi\in\mathcal{V},\ \textit{for all}\ \ \pi\in\Pi\ \ \mathrm{ex}(\varphi,\pi)\in|\bot|^\sigma\}$$

We note that $\varphi\in\overline{\Pi}$ implies that $\mathrm{ex}(\varphi,\pi)$ is well defined for all $\pi\in\Pi$.

It is very important to note that the definition of $\overline{\Pi}$ depends of $|\bot|^\sigma$ which is in fact the set of all legal stacks when a program stops.

Given such an interpretation, we define by induction the interpretation for all first order terms, stack formulas and value formulas. The interpretation of a term is an element of $\mathcal{V}$, the interpretation of a stack formula is an element of $\mathcal{P}(\mathcal{S})$ and the interpretation of a value formula is an element of $\mathcal{P}(\mathcal{V})$:

$$\begin{array}{rcl}
|f(t_1,\ldots,t_n)|^\sigma & = & |f|^\sigma(|t_1|^\sigma,\ldots,|t_n|^\sigma) \\
|X(t_1,\ldots,t_n)|^\sigma & = & |X|^\sigma(|t_1|^\sigma,\ldots,|t_n|^\sigma) \\
|A(t_1,\ldots,t_n)|^\sigma & = & |A|^\sigma(|t_1|^\sigma,\ldots,|t_n|^\sigma) \\
|\top|^\sigma & = & \{\varepsilon\} \\
|\neg P|^\sigma & = & \overline{|P|^\sigma} \\
|F \wedge P|^\sigma & = & |F|^\sigma \times |G|^\sigma \\
|\exists X P|^\sigma & = & \bigcup_{\Pi\in\mathcal{V}^n\to\mathcal{P}(\mathcal{S})} |P|^{\sigma[\Pi/X]} \quad (n \text{ is the arity of } X) \\
|\exists A P|^\sigma & = & \bigcup_{\Phi\in\mathcal{V}^n\to\mathcal{P}(\mathcal{V})} |P|^{\sigma[\Phi/A]} \quad (n \text{ is the arity of } A) \\
|\exists x P|^\sigma & = & \bigcup_{\varphi\in\mathcal{V}'} |P|^{\sigma[\varphi/x]}
\end{array}$$

We introduce also the following notation:

- If $\Gamma = F_1,\ldots,F_n$, we will denote $|\Gamma|^\sigma = \{(\varphi_1,\ldots,\varphi_n), \varphi_i \in |F_i|^\sigma\}$.
- For any interpretation σ, we will denote $\sigma[\Phi/\chi]$ for the usual modification of the interpretation of only one variable.
- To deal with second order, we will also denote $|\lambda x_1 \ldots \lambda x_n\, M|^\sigma$ (with $M \in \mathcal{F}^\Phi$ or $M \in \mathcal{F}^\Pi$) for the function defined $\varphi_1,\ldots,\varphi_n \mapsto |M|^{\sigma[\varphi_1/x_1]\ldots[\varphi_n/x_n]}$.

Proposition 4.4 *We have the usual proposition: for all interpretation σ, for all value formula F, and for all stack formula P, we have:*

$$\begin{array}{rcl}
|F[\varphi/\chi]| & = & |F|^{\sigma[|\varphi|^\sigma/\chi]} \\
|P[\varphi/\chi]| & = & |P|^{\sigma[|\varphi|^\sigma/\chi]}
\end{array}$$

proof: The proof is done by induction on the formulas F and P. ♠

5 The type system.

We can also use proof to associate programs to formulas. As in the Krivine's and Leivant's system AF_2, we choose a set $\mathcal{E}$ of equational axioms on first order terms and add a rule for these equations. Here are the rules with their algorithmic contents:

$$\frac{}{\mathtt{Jump}_m : F_1,\ldots,F_{m-1},\neg P,F_{m+1},\ldots,F_n \mid P \vdash}\, Ax^\Gamma \qquad \frac{p : F_1,\ldots,F_n \mid (F_m \wedge P) \vdash}{\mathtt{Push}_m;p : F_1,\ldots,F_n \mid P \vdash}\, Co$$

$$\frac{p : F_1,\ldots,F_n \mid P \vdash}{\mathtt{Pop}_n;p : \Gamma \mid F_1 \wedge \ldots \wedge F_n \wedge P \vdash}\, \wedge_i \qquad \frac{p : \Gamma \mid \neg Q \wedge P \vdash \quad q : \Gamma \mid Q \vdash}{\mathtt{Push}[q];p : \Gamma \mid P \vdash}\, \wedge\neg_e$$

$$\frac{p : \Gamma \mid \top \vdash}{\mathtt{Erase};p : \Gamma \mid P \vdash}\, \top_e \qquad \frac{p : \Gamma \mid \neg(\neg P \wedge \top) \wedge P \vdash}{\mathtt{Save};p : \Gamma \mid P \vdash}\, \top'_e$$

$$\frac{p : \Gamma \mid P \vdash \qquad \chi \notin \Gamma}{p : \Gamma \mid \exists \chi\, P \vdash}\ \exists_i \qquad \frac{p : \Gamma \mid \exists \chi\, P \vdash}{p : \Gamma \mid P[\varphi/\chi] \vdash}\ \exists_e$$

$$\frac{}{\mathtt{Stop} : \Gamma \mid \bot \vdash}\ \bot_s \qquad \frac{p : \Gamma \mid P[t/x] \vdash \quad \mathcal{E} \vdash t = u}{p : \Gamma \mid P[u/x] \vdash}\ Eq$$

Proposition 5.1 *Let σ be an interpretation verifying: $\mathcal{E} \vdash t = u$ implies $|t|^\sigma = |u|^\sigma$ for all first order terms. If we prove $p : \Gamma \mid P \vdash$ then for all $e \in |\Gamma|^\sigma$, we have*

$$\langle p/e \rangle \in |\neg P|^\sigma$$

proof: We prove this result by induction on the proof:

- If the last rule is Ax^Γ:

$$\frac{}{\mathtt{Jump}_m : F_1, \ldots, F_{m-1}, \neg P, F_{m+1}, \ldots, F_n \mid P \vdash}\ Ax^\Gamma$$

  Let $e = (\varphi_1, \ldots, \varphi_n) \in \mathcal{V}$ be such that $\varphi_i \in |F_i|^\sigma$. We have $\mathrm{ex}(\langle \mathtt{Jump}_m/e \rangle, \pi) = \mathrm{ex}(\langle p'/e' \rangle, \pi)$ with $\varphi_m = \langle p'/e' \rangle$. So $\langle \mathtt{Jump}_m/e \rangle \in |\neg P|^\sigma$, because $\varphi_m \in |\neg P|^\sigma$.

- If the last rule is Co

$$\frac{p : F_1, \ldots, F_m, \ldots F_n \mid (F_m \wedge P) \vdash}{\mathtt{Push}[m]; p : F_1, \ldots, F_m, \ldots F_n \mid P \vdash}\ Co$$

  We denote $\Gamma = F_1, \ldots, F_n$. We have to prove $\langle \mathtt{Push}_m; p/e \rangle \in |\neg P|^\sigma$. We choose $e \in |\Gamma|^\sigma$ and $\pi \in |P|^\sigma$. By definition, we have $e = \langle \varphi_1 \ldots \varphi_m \ldots \varphi_n \rangle$ with for all i, $\varphi_i \in |F_i|^\sigma$. Thus, if we denote $\pi' = \varphi_m \cdot \pi$ we get $\mathrm{ex}(\langle \mathtt{Push}_m; p/e \rangle, \pi) = \mathrm{ex}(\langle p/e \rangle, \pi')$. Hence, we get the expected result using $\langle p/e \rangle \in |\neg(F_m \wedge P)|^\sigma$ (by induction hypothesis) and $\pi' \in |F_m \wedge P|^\sigma$.

- If the last rule is $\wedge_i$

$$\frac{p : F_1, \ldots, F_n \mid P \vdash}{\mathtt{Pop}_n; p : \Gamma \mid F_1 \wedge \ldots \wedge F_n \wedge P \vdash}\ \wedge_i$$

  We denote $\Gamma' = F_1, \ldots, F_n$. We have to prove $\langle \mathtt{Pop}_n; p/e \rangle \in |\neg(F_1 \wedge \ldots \wedge F_n \wedge P)|^\sigma$ for all $e \in |\Gamma|^\sigma$. We choose $e \in |\Gamma|^\sigma$ and $\pi \in |F_1 \wedge \ldots \wedge F_n \wedge P|^\sigma$. By definition, we have $\pi = \varphi_1 \cdots \varphi_n \cdot \pi'$ with for all i, $\varphi_i \in |F_i|^\sigma$ and $\pi' \in |P|^\sigma$. Then, $\mathrm{ex}(\langle \mathtt{Pop}_n; p/e \rangle, \pi) = (\langle p/e' \rangle, \pi')$ with $e' = (\varphi_1, \ldots, \varphi_n)$, and we get the result using $e' \in |\Gamma'|^\sigma$, $\langle p/e' \rangle \in |\neg P|^\sigma$ (induction hypothesis) and $\pi' \in |P|^\sigma$.

- If the last rule is $\wedge\neg_e$

$$\frac{p : \Gamma \mid \neg Q \wedge P \vdash \quad q : \Gamma \mid Q \vdash}{\mathtt{Push}[q]; p : \Gamma \mid P \vdash}\ \wedge\neg_e$$

  We have to prove $\langle \mathtt{Push}[q]; p/e \rangle \in |\neg P|^\sigma$. We choose $e \in |\Gamma|^\sigma$ and $\pi \in |P|^\sigma$. We have $\mathrm{ex}(\langle \mathtt{Push}[q]; p/e \rangle, \pi) = \mathrm{ex}(\langle p/e \rangle, \langle q/e \rangle \cdot \pi)$. Thus, we get the result because by induction hypothesis we have $\langle p/e \rangle \in |\neg(\neg Q \wedge P)|^\sigma$ and $\langle q/e \rangle \in |\neg Q|^\sigma$, so we have $\langle q/e \rangle \cdot \pi \in |\neg Q \wedge P|^\sigma$.

- If the last rule is $\top_e$

$$\frac{p : \Gamma \mid \top \vdash}{\texttt{Erase};p : \Gamma \mid P \vdash}\ \top_e$$

Let $e \in |\Gamma|^\sigma$ be, we have to prove $\langle \texttt{Erase};p/e \rangle \in |\neg P|^\sigma$. By definition, for all $\pi \in |P|^\sigma$, we have $\mathrm{ex}(\langle \texttt{Erase};p/e \rangle, \pi) = \mathrm{ex}(\langle p/e \rangle, \varepsilon)$. Hence, we get the wanted result because $\langle p/e \rangle \in |\neg \top|^\sigma$ (induction hypothesis).

- If the last rule is $\bot_s$

$$\frac{}{\texttt{Stop} : \Gamma \mid \bot \vdash}\ \bot$$

For all $e \in |\Gamma|^\sigma$ and $\pi \in |\bot|^\sigma$, we have $\mathrm{ex}(\langle \texttt{Stop}/e \rangle, \pi) = \pi \in |\bot|^\sigma$. So we have $\langle \texttt{Stop}/e \rangle \in |\neg \bot|^\sigma$.

- If the last rule is $\top_c$

$$\frac{p : \Gamma \mid \neg(\neg P \wedge \top) \wedge P \vdash}{\texttt{Save};p : \Gamma \mid P \vdash}\ \top_c$$

Let $e \in |\Gamma|^\sigma$ be, we have to prove that $\langle \texttt{Save};p/e \rangle \in |\neg P|^\sigma$. Let $\pi \in |P|^\sigma$ be, so we have $\mathrm{ex}(\langle \texttt{Save};p/e \rangle, \pi) = \mathrm{ex}(\langle p/e \rangle, \varphi \cdot \pi)$, with $\varphi = \langle \texttt{Rest}/\pi \rangle$. If we prove $\varphi \in |\neg(\neg P \wedge \top)|^\sigma$, we get $\varphi \cdot \pi \in |\neg(\neg P \wedge \top) \wedge P|^\sigma$ and with the induction hypothesis, we get the wanted result.

Now let us prove that $\varphi \in |\neg(\neg P \wedge \top)|^\sigma$. Let us choose $\pi' \in |\neg P \wedge \top|^\sigma$. By definition, $\pi' = \langle p'/e' \rangle \cdot \varepsilon$ with $\langle p'/e' \rangle \in |\neg P|^\sigma$. But we have $\mathrm{ex}(\langle \texttt{Rest}/\pi \rangle, \pi') = \mathrm{ex}(\langle p'/e' \rangle, \pi)$. Hence we get the expected result because $\langle p'/e' \rangle \in |\neg P|^\sigma$ and $\pi \in |P|^\sigma$.

- If the last rule is $\exists_i$

$$\frac{p : \Gamma \mid P \vdash \qquad \chi \notin \Gamma}{p : \Gamma \mid \exists \chi\, P \vdash}\ \exists_i$$

Let σ be an interpretation and choose $e \in |\Gamma|^\sigma$, we have to prove that $\langle p/e \rangle \in |\neg \exists \chi\, P|^\sigma$. But this signify that for all possible interpretation φ for the variable χ, we have $\langle p/e \rangle \in \overline{|P|^{\sigma[\varphi/\chi]}}$, and this is true by induction hypothesis.

- If the last rule is $\exists_e$

$$\frac{p : \Gamma \mid \exists \chi\, P \vdash Q}{p : \Gamma \mid P[\varphi/\chi] \vdash Q}\ \exists_e$$

By induction hypothesis, for all interpretation σ and for all $e \in |\Gamma|^\sigma$, $\langle p/e \rangle \in |\neg \exists \chi\, P|^\sigma$, so we have $\langle p/e \rangle \in \overline{|P|^{\sigma[|\varphi|^\sigma/\chi]}}$, for all expression φ substitutable to χ, and by the proposition 4.4, we have $|P[\varphi/\chi]|^\sigma = |P|^{\sigma[|\varphi|^\sigma/\chi]}$. Hence we get $\langle p/e \rangle \in |\neg P[\varphi/\chi]|^\sigma$.

- If the last rule is Eq

$$\frac{p : \Gamma \mid P[t/x] \vdash \quad \mathcal{E} \vdash t = u}{p : \Gamma \mid P[u/x] \vdash}\ Eq$$

By hypothesis on σ and by 4.4, we get $|P[t/x]|^\sigma = |P[u/x]|^\sigma$. So we get the wanted result.

♠

Definition 5.2 *We will say that M is a control formula, if $|M|^\sigma$ is independent of the interpretation σ (we can apply this definition if M is a stack or a value formula). In this case we note $|M|$ the interpretation of M.*

We can see that $\top$ is a control stack formula.

Proposition 5.3 *Let $F_1, \ldots, F_n$ be value control formulas and P,Q control stack formulas. If we prove $p : F_1, \ldots, F_n \mid \neg Q \wedge P \vdash$, then for all $\pi \in |P|$ and $e \in |F_1, \ldots, F_n|$, we have $\mathrm{ex}(\langle p/e \rangle, \varphi \cdot \pi) \in |Q|$, with $\varphi = \langle \mathtt{Stop}/() \rangle$.*

proof: Let $F_1, \ldots, F_n$ be value control formulas and P, Q control stack formulas. We assume $p : F_1, \ldots, F_n \mid \neg Q \wedge P \vdash$. We can choose an interpretation σ such that $|\bot|^\sigma = |Q|$. So we get $\varphi = \langle \mathtt{Stop}/() \rangle \in |\neg Q|$, and we can apply the proposition 5.1, and we get $\langle p/e \rangle \in |\neg(\neg Q \wedge P)|$ for all $e \in |F_1, \ldots, F_n|$. So we have $\mathrm{ex}(\langle p/e \rangle, \varphi \cdot \pi) \in |\bot|^\sigma = |Q|$ for all $\pi \in |P|$. ♠

6 Data types.

It is easy to add data types in this system. Let us show how to add integers. This example is demonstrative enough to show how to do with any kind of data types.

To add Integers, we follow these steps:

- We add a second order value constant of arity one: $N(\_)$. We will use the N symbol without parenthesis to simplify writing. We add the formula Nt to the set of value formulas.
- We add the following symbols to the language: 0, s(_), add(_,_), mul(_,_) .... We will use the s symbol without parenthesis to simplify writing. We identify all the elements of the data type with the set of logical terms obtain with its constructors. For integer, this means that the element of the data type N are the logical terms of the form $s^n 0$.
- We add to the set of equations $\mathcal{E}$ some equations to define add(_,_), mul(_,_) ....
- We add all the integers to the set of Value $\mathcal{V}$. So we can put some integers in the the stack or in the environment. We will denote $\overline{i}$ the value associated to an element i of the data type N.
- We add $\mathtt{Push}_{\mathbb{N}}[i]$ (for each integer i), $\mathtt{Rec}_{\mathbb{N}}[p_0][p_s]$ (for each program p_0 and p_s) and $\mathtt{Inc}_{\mathbb{N}}$ to the set of instructions $\mathcal{I}$ and we gives the following transition tables:

input code output code	input environment output environment	input stack output stack
$\mathtt{Push}_{\mathbb{N}}[i];p$ p	e e	π $\overline{i} \cdot \pi$
$\mathtt{Inc}_{\mathbb{N}};p$ p	e e	$\overline{i} \cdot \pi$ $\overline{i+1} \cdot \pi$
$\mathtt{Rec}_{\mathbb{N}}[p_0][p_s]$ p_0	$(\overline{0})$ $()$	π π
$\mathtt{Rec}_{\mathbb{N}}[p_0][p_s]$ p_s	$(\overline{i+1})$ $(\overline{i})$	π $\langle \mathtt{Rec}_{\mathbb{N}}[p_0][p_s]/(\overline{i}) \rangle \cdot \pi$

- We add the following rules to the type system:

$$\frac{p : \Gamma \mid N(i) \wedge P \vdash}{\mathtt{Push}_{\mathbb{N}}[i];p : \Gamma \mid P \vdash}\,\mathbb{N}_{ex} \qquad \frac{p : \Gamma \mid N(\mathrm{s}\, i) \wedge P \vdash}{\mathtt{Inc}_{\mathbb{N}};p : \Gamma \mid N(i) \wedge P \vdash}\,\mathbb{N}_{s}$$

$$\frac{p_0 : \mid P(0) \vdash \qquad p_s : N(j) \mid (\neg P(j) \wedge P(\mathrm{s}\, j)) \vdash}{\mathtt{Rec}_{\mathbb{N}}[p_0][p_s] : N(i) \mid P(i) \vdash}\,\mathbb{N}_{r}$$

- And finally we extend the notion of interpretation by $|N|^{\sigma}(i) = \{\bar{i}\}$ if i is an element of the data type N and $|N|^{\sigma}(u) = \emptyset$ in all the other cases. With this definition it is not difficult to extend the proof of proposition 5.1.

As an example, here is a derivation for the addition program, using the usual equation for addition, $\mathrm{add}(0, j) = j$ and $\mathrm{add}(\mathrm{s}\, i, j) = \mathrm{s}\, \mathrm{add}(i, j)$. To simplify the proof we use this notation:

$$\begin{array}{rcl} P^j(i) & = & \neg(N(\mathrm{add}(i, j)) \wedge Q) \wedge N(j) \wedge Q \\ \Gamma^j(i) & = & \neg P^j(i), \neg(N(\mathrm{add}(\mathrm{s}\, i, j)) \wedge Q) \end{array}$$

Here is the proof:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\neg(N\,\mathrm{add}(0,j)\wedge Q) \mid N\,\mathrm{add}(0,j)\wedge Q \vdash}\,Ax^{\Gamma}}{\neg(N\,\mathrm{add}(0,j)\wedge Q) \mid Nj \wedge Q \vdash}\,Eq}{\mid P^j(0) \vdash}\,\wedge_i \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma^j(k) \mid P^j(k) \vdash}\,Ax^{\Gamma} \qquad \dfrac{\dfrac{\dfrac{}{\Gamma^j(k) \mid N\,\mathrm{add}(\mathrm{s}\,k,j)\wedge Q \vdash}\,Ax^{\Gamma}}{\Gamma^j(k) \mid N\,\mathrm{s}\,\mathrm{add}(k,j)\wedge Q \vdash}\,Eq}{\Gamma^j(k) \mid N\,\mathrm{add}(k,j)\wedge Q \vdash}\,\mathbb{N}_s}{\Gamma^j(k) \mid Nj \wedge Q \vdash}\,\neg_e}{Nk \mid \neg P^j(k) \wedge P^j(\mathrm{s}\,k) \vdash}\,\wedge_i}{Ni \mid P^j(i) \vdash}\,\mathbb{N}_r}{Ni, Nj, \neg(N\,\mathrm{add}(i,j)\wedge Q) \mid Nj \wedge Q \vdash}\,Ax^{\Pi}}{Ni, Nj, \neg(N\,\mathrm{add}(i,j)\wedge Q) \mid Q \vdash}\,Ax^{\Pi}}{\mid Ni \wedge Nj \wedge \neg(N\,\mathrm{add}(i,j)\wedge Q) \wedge Q \vdash}\,\wedge_i$$

And the program extracted from this proof:

$$\mathtt{Pop}_3;\mathtt{Push}_2;\mathtt{Push}_3;\mathtt{Rec}_{\mathbb{N}}[\mathtt{Pop}_1;\mathtt{Jump}_1][\mathtt{Pop}_2;\mathtt{Push}[\mathtt{Inc}_{\mathbb{N}};\mathtt{Jump}_2];\mathtt{Jump}_1]$$

We may comment on this proof:

- The conclusion of the proof: $\mid Ni \wedge Nj \wedge \neg(N\,\mathrm{add}(i,j)\wedge Q) \wedge Q \vdash$ is a type for an addition program in continuation passing style. This program waits, on the stack, for two integers of type $N(i)$ and $N(j)$, a "continuation" of type $\neg(N\,\mathrm{add}(i,j)\wedge Q)$ and the rest of the stack of type Q. Then, this program computes the sum of i and j, and calls the continuation .
- We may distinguish three parts in this proof, separated by the recursion rule: The 0 and the successor case and, at the bottom of the proof, some manipulations on the stack to put things in the right order.

7 Correctness and optimizations.

We are going to show how the use of data types as above implies the correctness of the program. In fact we use the fact that by definition data types are control formulas. We can write this proposition:

Proposition 7.1 *If $D_0, D_1, \ldots, D_n$ are some data types, if f is a function symbol of arity n and if we can find a model for our equations $\mathcal{E}$ (this means that our equations don't implies that some distinct elements of a data type are equal). Then if we have a proof of*

$$p : \mid D_1(x_1) \wedge \ldots \wedge D_n(x) \wedge \neg(D_0(f(x_1, \ldots, x_n)) \wedge \top) \wedge \top \vdash$$

the program p is a program for the function f in the sense that for all $i_1, \ldots, i_n$ respectively elements of the data types $D_1, \ldots, D_n$, we have

$$\mathrm{ex}(\langle p/() \rangle, \overline{i_1} \cdots \overline{i_n} \cdot \varphi \cdot \varepsilon) = \overline{\Phi(i_1 \cdot \cdots \cdot i_n)} \cdot \varepsilon \quad \textit{with} \quad \varphi = \langle \mathtt{Stop}/() \rangle$$

where Φ is a function from $D_1, \ldots, D_n$ to D_0 verifying for all $i_0, i_1, \ldots, i_n$ in $D_0, D_1, \ldots, D_n$, $\mathcal{E} \vdash f(i_1, \ldots, i_n) = i_0$ implies $\Phi(i_1, \ldots, i_n) = i_0$.

proof: First, we remark that if we can find a model for our equations $\mathcal{E}$, then we can find an interpretation σ such that for all first order terms, $\mathcal{E} \vdash u = v$ implies $|u|^\sigma = |v|^\sigma$. To prove that we use standard method to extend the model defined only on the data to the set of all values.

After this, we may apply the proposition 5.3 to the previous proof (because $D_0, D_1, \ldots, D_n$ and $\top$ are control formula). We get

$$\mathrm{ex}(\langle p/() \rangle, \overline{i_1} \cdots \overline{i_n} \cdot \varphi \cdot \pi) \in |D_0(f(x_1, \ldots, x_n)) \wedge Q|^{\sigma[x_1/\overline{i_1}]\ldots[x_n/\overline{i_n}]}$$

Thus by definition of the interpretation σ, we can define the function Φ with $\overline{\Phi(i_1, \ldots, i_n)} \in |D_0(f(i_1, \ldots, i_n))|^\sigma$ and

$$\mathrm{ex}(\langle p/() \rangle, \overline{i_1} \cdots \overline{i_n} \cdot \varphi \cdot \varepsilon) = \overline{\Phi(i_1 \cdot \cdots \cdot i_n)} \cdot \varepsilon$$

But, by definition of the interpretation of a data type, there is an unique element in $|D_0(f(i_1, \ldots, i_n))|^\sigma$. Hence we have $|f(i_1, \ldots, i_n)|^\sigma = \overline{\Phi(i_1, \ldots, i_n)}$. So $\mathcal{E} \vdash f(i_1, \ldots, i_n) = i_0$ implies $\Phi(i_1, \ldots, i_n) = i_0$.

♠

To prove that a program is correct we use only the proposition 5.1 and its corollary 5.3. Then, we can give a very general definition of optimization which preserve these properties.

First we remark that if we add some instructions to the machine (by adding elements to the set $\mathcal{I}$), because the system doesn't use them, all the previous results are preserved. Now, if we consider a derived rule:

$$\frac{p : \Gamma \mid P \vdash Q}{p' : \Gamma' \mid P' \vdash Q'}$$

We may use some new instructions to produce a better programs p''. We preserve the result of the proposition 5.1 using p'' instead of p' if for all interpretation σ, all $e \in |\Gamma|^\sigma$ and all $e' \in |\Gamma'|^\sigma$ we have $\langle p/e \rangle \in |P \to Q|^\sigma$ implies $\langle p''/e' \rangle \in |P' \to Q'|^\sigma$.

In this case we will say that the following rule is an optimized rule:

$$\frac{p : \Gamma \mid P \vdash Q}{p'' : \Gamma' \mid P' \vdash Q'}$$

An important remark is that we don't impose that p' and p'' be "equivalent" programs in any sense. In fact p' and p'' can do totally different things.

This notion of derived rules says that p'' is compatible with the notion of interpretation, this means that it do the same thing than p' only if used with data types in a "well typed environment".

8 Call by name compilation of system AF_2.

The principle of this compilation is to translate a proof from system AF_2 in MD_{SEC} (You can find the definition of system AF_2 in annex A). The first thing is to remark that we can replace the original introduction of implication rule by this multiple introduction:

$$\frac{M_1, \ldots, M_n \vdash_{\mathrm{AF}_2} M}{\Gamma \vdash_{\mathrm{AF}_2} M_1 \to (M_2 \to \ldots (M_n \to M))} \to_i^n$$

The translation of a proof in order to use this rule instead of the original one is in fact the lambda lifting. This first translation is left to the reader. (We need to put a non empty context in the conclusion of the rule because there is no weakening in the system).

Now we translate the formula of system AF_2:

Definition 8.1 *We define by induction on a formula M of system AF_2, a formula $M^0 \in \mathcal{F}^{\Pi}$, and $\overline{M} \in \mathcal{F}^{\Phi}$ by*

- $\overline{M} = \neg M^0$
- $X(\overline{t})^0 = X(\overline{t})$ *where* $X \in \mathcal{V}^{\Pi}$
- $(M \to N)^0 = \overline{M} \wedge N^0$
- $(\forall x\, M)^0 = \exists x\, M^0$
- $(\forall X\, M)^0 = \exists X\, M^0$

Note: All the variables are translated in stack variables, so we identify the set of variables of system AF_2 to the set of stack variables.

Proposition 8.2 *For all formulas M and N of system AF_2 and all variable X of arity n, we have*

$$(M[\lambda x_1 \ldots \lambda x_n N / X])^0 = M^0[\lambda x_1 \ldots \lambda x_n N^0 / X]$$

proof: We prove this by induction on the formula M. ♠

Now it is possible to translate the proof (For any environment $\Gamma = M_1, \ldots, M_n$ we will note $\overline{\Gamma} = \overline{M_1}, \ldots, \overline{M_n}$):

Proposition 8.3 *For all environment* Γ *and all formula* M *of system* AF_2,

$$\Gamma \vdash_{\mathrm{AF}_2} M \quad \textit{implies} \quad \overline{\Gamma} \mid M^0 \vdash$$

proof: We prove this by induction on the proof of $\Gamma \vdash_{\mathrm{AF}_2} M$:

- If the last rule is an axiom: We have $M = M_i$ with $\Gamma = M_1, \ldots, M_n$. Hence, we get the expected result using the Ax^Γ rule, because $\overline{M_i} = \neg M_i^0$.
- If the last rule is the elimination of implication, by induction hypothesis, we get $\Gamma \vdash_{\mathrm{AF}_2} N \to M$ implies $\overline{\Gamma} \mid \neg N^0 \wedge M^0 \vdash$ and $\Gamma \vdash_{\mathrm{AF}_2} N$ implies $\overline{\Gamma} \mid N^0 \vdash$. Hence, we get the expected result using the $\wedge\neg_e$ rule.
- If the last rule is the multiple introduction of implication, we have $M = N_1 \to (N_2 \to \ldots(N_p \to N))$ and by induction hypothesis we get $N_1, \ldots, N_p \vdash_{\mathrm{AF}_2} N$ implies $\overline{N_1}, \ldots, \overline{N_p} \mid N^0 \vdash$. Hence, we get the expected result using the $\wedge_i$.
- If the last rule is the elimination of a universal quantifier, we have $M = N[\varphi/\chi]$. By induction hypothesis we get $\Gamma \vdash_{\mathrm{AF}_2} \forall\chi\, N$ implies $\overline{\Gamma} \mid \exists\chi\, N^0 \vdash$. If χ is a first order variable then φ is a term and $M^0 = N^0[\varphi/\chi]$. If χ is a second order variable then we get $M^0 = N^0[\varphi^0/\chi]$ by proposition 8.2. Hence, we have the expected result using the $\exists_e$ rule.
- If the last rule is the introduction of a universal quantifier, we have $M = \forall\chi\, N$ and by induction hypothesis we get $\Gamma \vdash_{\mathrm{AF}_2} N$ implies $\overline{\Gamma} \mid N^0 \vdash$. Moreover, we know that χ is not free in $\overline{\Gamma}$. Hence, we get the expected result using the $\exists_i$ rule.
- If the last rule is the equational rule, the induction hypothesis and the rule Eq rule give directly the expected result.

♠

Now, if we compare the term extracted from the proof in natural deduction (with the multiple implication introduction rule) and the code extracted from the translated proof, we obtain the following compilation for the lambda-calculus using super-combinator (we use $\overline{t}$ to denote the code of a term t):

- $x_i \mapsto \mathtt{Jump}_i$ (axiom rules are translated with the Ax^Γ rule)
- $\lambda x_1 \ldots \lambda x_n\, t \mapsto \mathtt{Pop}_n;\overline{t}$ (multiple implication introduction rule are translated with the $\wedge_i$ rule)
- $(t\ u) \mapsto \mathtt{Push}[\overline{u}];\overline{t}$ (implication elimination rule are translated with the $\wedge\neg_e$ rule)
- Nothing more, because all other rules of system AF_2 have no algorithmic contents and are translated in non-algorithmic rules of MD_{SEC}.

This sounds like a good call by name compilation. Moreover, one can easely obtain usual optimization. For instance we can translate an elimination of implication on an axiom with the Co rule instead of using $\wedge\neg_e$ and Ax^Γ rules. So we get a well known optimization which gives $(t\ x_i) \mapsto \mathtt{Push}_i;\overline{t}$ instead of $(tx_i) \mapsto \mathtt{Push}[\mathtt{Jump}_i];\overline{t}$.

Moreover, if we add to system AF_2 the $\bot$ formula with both intuitionist or classical absurdity:

$$\frac{\Gamma \vdash_c t : \bot}{\Gamma \vdash_c (A\ t) : M} \qquad \frac{\Gamma \vdash_c t : (M \to \bot) \to M}{\Gamma \vdash_c (C\ t) : M}$$

We can always translate the proof: we extend the translation of formulas with $\bot^0 = \top$ (we get $((M \to \bot) \to M)^0 = \neg(\neg M^0 \wedge \top) \wedge M^0$. The two previous rules are translated using the $\top_e$ and $\top_c$. This gives the following compilation for the A and C operator: $(A\ t) \mapsto \texttt{Erase};\overline{t}$ and $(C\ t) \mapsto \texttt{Save};\overline{t}$. This is a possible compilation for the following reduction rules (in call-by-name) for these operators:

$$\begin{array}{lcl} (A\ t\ t_1 \ldots t_n) & \triangleright & t \\ (C\ t\ t_1 \ldots t_n) & \triangleright & (t\ \lambda x\ (x\ t_1 \ldots t_n)\ t_1 \ldots t_n) \end{array}$$

Corollary 8.4 *We have:* $\Gamma \vdash_{\mathrm{AF}_2} M$ *in system* AF_2 *with the previous rule added if and only if* $\overline{\Gamma} \mid M^0 \vdash$ *in* MD_{SEC}.

proof: The left-right implication is a consequence of the previous translation. The right-left implication is easy to prove and left to the reader (it's a consequence of the proposition 2.2). ♠

9 Conclusion and further outlook.

This new type system shows how it is possible to use the proof as program paradigm for something really different from the lambda-calculus: some code for an abstract machine.

Moreover, it's a good system to translate second order classical logic into intuitionistic logic with an explicit algorithmic content. In fact, we could see a relation between a kind of A-translation and the call-by-name compilation.

A question is to know how this depends on the type system. I have also design another type system for an S.E.C.D. machine (In fact this is the original type system I have found and the actual S.E.C. version is simply a restriction of this system as the machine is). I have proved this system complete for classical logic ! In fact there is only one rule which give classical logic: a rule to save the dump (like the $\top'_e$ rule in MD_{SEC} is used to save the stack). Hence the relations between classical logic and compilation for an abstract machine seems not so clear.

Another problem is the unsymmetrical character of the system: we can only add instructions to the beginning of a program. Hence, some compilations are not possible. For instance, it's possible to compile call-by-value but you don't get the expected code. (because one of the natural compilation of $(u\ v)$ is $\overline{u};\overline{v};\texttt{Apply}$, so you need to be able to add instructions to the both sides of $\overline{v}$). It seems not to difficult to write a symmetrical system with both left and right rules to add instructions respectively to the beginning and to the end of a program, and a true cut rule to compose two programs. This kind of system could be powerful enough to really reach most of the compilations for a given machine.

The last possible direction for further work is about optimization. Because we use only the notion of interpration to ensure the correctness, we have more facilities to manipulate the code. Moreover, because we have a lot of information inside the proof we could expect a lot of opimizations which are usually very difficult or impossible to do. One possibility could be to save only the position of the stack when we use this feature only as an exception mechanism (this means when we restore the stack only "inside" the procedure which has saved it). Another possibility could be to optimize a program automaticaly using some physical manipulation. This direction of work is perhaps very promising but is not so easy.

A Definition of system AF_2.

System AF_2 uses classical second order formulas construct with first order terms from a language $\mathcal{L}$, atomic formulas, implication, second order and first order universal quantification $(X(t_1,\ldots,t_n) \mid F \to G \mid \forall X\, F \mid \forall x\, F)$.

It uses also a set of equations on first order terms $\mathcal{E}$. Sequent are of the form $x_1 : F_1, \ldots, x_n : F_n \vdash t : F$ where $x_1, \ldots, x_n$ are lambda-variables and where t is a lambda-term (whose free variables are among $x_1, \ldots, x_n$). This sequent may be proved using the following rules:

- Axiom and equational rules:

$$\frac{}{x : F, \Gamma \vdash x : F}\, Ax \qquad \frac{\Gamma \vdash t : F\,[x/u] \qquad E \vdash u = v}{\Gamma \vdash t : F\,[x/v]}\, Eq$$

- Implication rules:

$$\frac{x : F, \Gamma \vdash t : G}{\Gamma \vdash \lambda x\, t : F \to G} \to_i \qquad \frac{\Gamma \vdash t : F \to G \qquad \Gamma \vdash u : F}{\Gamma \vdash (t)u : G} \to_e$$

- First order abstraction rules:

$$\frac{\Gamma \vdash t : F \qquad \chi \notin \Gamma}{\Gamma \vdash t : \forall \chi F} \forall_i \qquad \frac{\Gamma \vdash t : \forall \chi F}{\Gamma \vdash t : F[\varphi/\chi]} \forall_e$$

The reader could find more information about system AF_2 in [9], [7] and [6].

References

[1] H. P. Barendregt. *The Lambda Calculus: Its Syntax and Semantics*. North-Holland, revised edition, 1984.

[2] C. Böhm. Alcune proprieta delle forme $\beta\eta$-normali nel λk-calculus. Pubblicazioni 696, Instituto per le Applicazioni del Calcolo, Roma, 1968.

[3] J.-Y. Girard. The system F of variable types: fifteen years later. *Theoretical Computer Science*, 45:159–192, 1986.

[4] W. Howard. The formulae-as-types notion of construction. *To H.B. Curry: Essays on combinatory logic, λ-calculus and formalism*, pages 479–490, 1980.

[5] Jean-Louis Krivine. Opérateurs de mise en mémoire et traduction de Godël. Technical Report 3, Equipe de Logique de Paris 7, December 1989.

[6] Jean-Louis Krivine. *Lambda-Calcul : Types et Modèles*. Etudes et Recherches en Informatique. Masson, 1990. Available soon in english version.

[7] Jean-Louis Krivine and Michel Parigot. Programming with proofs. *Inf. Process. Cybern.*, EIK 26(3):149–167, 1990.

[8] Daniel Leivant. Typing and computational properties of lambda expressions. *Theoretical Computer Science*, 44:51–68, 1986.

[9] Michel Parigot. Programming with proofs: a second order type theory. *Lecture Notes in Computer Science*, 300, 1988. Communication at ESOP 88.

[10] Michel Parigot. Recursive programming with proofs. *Theoritical Computer Science*, 94:335–356, 1992.

[11] Simon L. Peyton J. *The Implementation of Functional Programming Languages*. Prentice-Hall, 1987. Prentice-Hall International Series in Computer Science.