A sequent calculus presentation of the Calculus of Inductive Constructions

(work in progress)

Hugo Herbelin

(jointly with Jeffrey Sarnat and Vincent Siles)

10 July 2010

Dependently Typed Programming Workshop

Edinburgh

Motivation

- sequent calculus can be seen as a λ-calculus

  $\hookrightarrow$ two main variants: LJ_T/LK_T for call-by-name, LJ_Q/LK_Q for call-by-value

- sequent calculus is a typing system for abstract machine, hence a priori for efficient reduction

  $\hookrightarrow$ left introduction rules build "stack", right introduction rules build code, cut rule builds states and closures

- sequent calculus is the natural framework for proof search

  $\hookrightarrow$ see e.g. Lengrand's presentation of Pure Type Systems in sequent calculus form

- sequent calculus is good at making some symmetries explicit

  $\hookrightarrow$ a symmetry syntactic presentation of fixpoints and cofixpoints and of the respective guard conditions

LJ_T aka Spine Calculus

LK_T and LK_Q (Danos, Joinet and Schellinx, 1995) are two dual complete restrictions of LK respectively connected to call-by-name and call-by-value λ-calculus with control

LJ_T is the intuitionistic restriction of LK_T

LJ_T normal proofs (unless LJ proofs) are in bijective correspondence with call-by-name normal λ-terms

LJ_T has been independently designed by Cervesato and Pfenning under the name of Spine Calculus

LJ_T aka Spine Calculus (the propositional case)

Two kinds of sequents: $\Gamma \vdash A$ and $\Gamma; B \vdash A$ (the place for B is called "stoup").

$$A \ ::= \ X \mid A \to A$$

$$\frac{}{\Gamma; A \vdash A}\ \textsc{Ax} \qquad \frac{\Gamma, A; A \vdash B}{\Gamma, A \vdash B}\ \textsc{Cont}$$

$$\frac{\Gamma \vdash A \qquad \Gamma; B \vdash C}{\Gamma; A \to B \vdash C}\ \to_L \qquad \frac{\Gamma, A \vdash B}{\Gamma \vdash A \to B}\ \to_R$$

$$\frac{\Gamma \vdash A \qquad \Gamma; A \vdash B}{\Gamma \vdash B}\ \textsc{Cut}$$

LJ_T aka Spine Calculus (the propositional case, annotated)

$$
\begin{array}{lcll}
M, N & ::= & xK \mid \lambda x : A.M \mid MK & (terms) \\
K, L & ::= & \epsilon \mid M :: K & (spines, or stacks)
\end{array}
$$

Two kinds of sequents:

$\Gamma \vdash M : A$ for terms
$\Gamma; A \vdash K : B$ for spines (expecting a term of type A for building a term of type B)

$$
\frac{}{\Gamma; A \vdash \epsilon : A}\ \textsc{Ax} \qquad \frac{(x : A) \in \Gamma \qquad \Gamma; A \vdash K : B}{\Gamma \vdash xK : B}\ \textsc{Cont}
$$

$$
\frac{\Gamma \vdash M : A \qquad \Gamma; B \vdash K : C}{\Gamma; A \to B \vdash M :: K : C}\ \to_L \qquad \frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x^A.M : A \to B}\ \to_R
$$

$$
\frac{\Gamma \vdash M : A \qquad \Gamma; A \vdash K : B}{\Gamma \vdash MK : B}\ \textsc{Cut}
$$

LJ_T aka Spine Calculus (the propositional case, annotated)

In LJ_T, the reduction rules are cut-elimination rules for an abstract machine.

code	*stack*		*next state or result*	
$(\lambda x^A.M)$	$N :: K$	$\rightarrow$	$M\{N/x\}$	K
(MK)	L	$\rightarrow$	M	$(K@L)$
$(\lambda x^A.M)$	ϵ	$\rightarrow$	$\lambda x^A.M$	
(xK)	L	$\rightarrow$	$x(K@L)$	

(we use here effective substitutions but it could be done with explicit ones)

LJ_T (the Pure Type Systems case, Lengrand, 2006)

$$\begin{array}{lcl} M, N, T, U & ::= & xK \mid \lambda x^T.M \mid MK \mid s \mid \Pi x^T.U \\ K & ::= & \epsilon \mid M :: K \end{array}$$

$$\frac{\Gamma \vdash T : s}{\Gamma; T \vdash \epsilon : T}\ \textsc{Ax} \qquad \frac{(x : T) \in \Gamma \qquad \Gamma; T \vdash K : U}{\Gamma \vdash xK : U}\ \textsc{Cont} \qquad \frac{\Gamma \vdash M : T \qquad \Gamma; T \vdash K : U}{\Gamma \vdash MK : U}\ \textsc{Cut}$$

$$\frac{\Gamma \vdash M : T \quad \Gamma; U\{M/x\} \vdash K : C \quad \Gamma \vdash \Pi x^T.U : s}{\Gamma; \Pi x^T.U \vdash M :: K : C}\ \rightarrow_L \qquad \frac{\Gamma, x : T \vdash M : U \quad \Gamma \vdash \Pi x^T.U : s}{\Gamma \vdash \lambda x^T.M : \Pi x^T.U}\ \rightarrow_R$$

$$\frac{\Gamma \text{ wf} \qquad (s, s') \in \mathcal{A}x}{\Gamma \vdash s : s'}\ \textsc{Sort} \qquad \frac{\Gamma \vdash T : s_1 \qquad \Gamma, x : T \vdash U : s_2 \qquad (s_1, s_2, s_3) \in \mathcal{R}el}{\Gamma \vdash \Pi x^T.U : s}\ \textsc{Pi}$$

$$\frac{\Gamma\{;C\} \vdash M : T \qquad \Gamma \vdash U : s \qquad T = U}{\Gamma\{;C\} \vdash M : U}\ \textsc{Conv}_R^1 \qquad \frac{\Gamma; T \vdash K : C \qquad \Gamma \vdash U : s \qquad T = U}{\Gamma; U \vdash K : C}\ \textsc{Co}$$

and same reduction rules

Adding inductive types

First introducing contexts and substitutions

To deal with the arity of inductive types and constructors, it is convenient to consider a "calculus of context" (see Pientka et al) with declarations asserting judgements:

$$\Gamma \; ++= \; \Gamma, x : [\Gamma \vdash T]$$

together with rules for defining substitutions:

$$\frac{}{\Gamma \vdash \epsilon : [\vdash T] \mapsto [\vdash T]} \qquad \frac{\Gamma \vdash M_0 : U_0 \qquad \Gamma \vdash \overrightarrow{M} : [\Gamma' \vdash T]\{M_0/x_0\} \mapsto [\vdash T']}{\Gamma \vdash M_0\overrightarrow{M} : [x_0 : U_0, \Gamma' \vdash T] \mapsto [\vdash T']}$$

and rules for applying these substitutions:

$$\frac{\Gamma \vdash N : [\Gamma' \vdash T] \qquad \Gamma \vdash \overrightarrow{M} : [\Gamma' \vdash T] \mapsto [\vdash T']}{\Gamma \vdash N\overrightarrow{M} : T'}$$

Adding dependent pattern-matching

We can then consider inductive types as declarations of the following form:

$$I : [\overrightarrow{z : V} \vdash s_I], C_i : [\overrightarrow{x_i : U_i} \vdash I\overrightarrow{N_i}]$$

and we interpret a case-analysis **match** N **with** $\ldots \mid C_i\overrightarrow{x_i} \rightarrow M_i \mid \ldots$ **end** of natural deduction as a cut between N and a continuation $[\ldots \mid C_i\overrightarrow{x_i} \rightarrow M_i \mid \ldots]$ that matches N. The extended syntax is:

$$\begin{array}{lll} M, N, T, U & {+}{+}{=} & C\overrightarrow{M} \mid I\overrightarrow{M} \\ K & {+}{+}{=} & [\ldots \mid C\overrightarrow{x} \rightarrow M \mid \ldots] \end{array}$$

Regarding typing, the placeholder is now dependent in types and we need to give it a name!

$$\frac{\Gamma, \overrightarrow{z : V}, y : I\overrightarrow{z} \vdash T : s \qquad \ldots \Gamma, \overrightarrow{x_i : U_i} \vdash M_i : T\{\overrightarrow{N_i}/\overrightarrow{z}\}\{C_i\overrightarrow{x_i}/y\} \ldots \qquad \ldots C_i : [\overrightarrow{x_i : U_i} \vdash I\overrightarrow{N_i}] \ldots}{\Gamma; y : I\overrightarrow{P} \vdash [\ldots \mid C_i\overrightarrow{x_i} \rightarrow M_i \mid \ldots] : T\{\overrightarrow{P}/\overrightarrow{x}\}\{y/y\}}$$

The reduction rule is

$$C_{i_0}(\overrightarrow{P})\ \ [\ldots \mid C_i\overrightarrow{x_i} \rightarrow M_i \mid \ldots] \quad \rightarrow \quad M_{i_0}\{\overrightarrow{P}/\overrightarrow{x_{i_0}}\}$$

Dependent cut

Because the typing rule for $[\ldots \mid C_i \overrightarrow{x_i} \to M_i \mid \ldots]$ is dependent in the type, the cut rule now needs to be dependent too:

$$\frac{\Gamma \vdash M : T \qquad \Gamma; y : T \vdash K : U}{\Gamma \vdash MK : U\{M/y\}} \ \textsc{Cut}$$

Adding fixpoints and cofixpoints

Adding fixpoints and cofixpoints

We want to exhibit a duality between fixpoints and cofixpoints. Let us first consider a tail-recursive fixpoint without dependencies at all:

$$f := \mathtt{fix}_f\ \lambda n.\mathtt{match}\ n\ \mathtt{with}\ 0\ \rightarrow\ 0 \mid S\,n\ \rightarrow\ f(S\,(S\,n))\ \mathtt{end}$$

Obviously, this function is cutting n with a continuation that does a case analysis on it, then depending on the result, recursively does the same case analysis. We want to interpret this recursive part as a fixpoint definition over evaluation contexts.

This suggests to consider variables α, β, ... for evaluation contexts as in

$$\begin{array}{lll} M, N, T, U & ++= & \mathtt{cofix}_x.M \\ K & ++= & \alpha \mid \mathtt{fix}_\alpha.K \end{array}$$

and to represent f above as the expression

$$\lambda n.\, n\,\mathtt{fix}_\alpha.[0 \rightarrow 0 \mid S\,n \rightarrow (S\,(S\,n))\alpha]$$

The reduction rules come naturally:

$$\begin{array}{lllll} M & \mathtt{fix}_\alpha.K & \rightarrow & M & K\{\mathtt{fix}_\alpha.K/\alpha\} \\ \mathtt{cofix}_x.M & K & \rightarrow & M\{\mathtt{cofix}_x.M/x\} & K \end{array}$$

Adding fixpoints and cofixpoints: typing

To type evaluation context variables, we need to consider sequents with several (non-dependent) conclusions, i.e. either of the form $\Gamma \vdash \Delta; M : T$ or $\Gamma; x : U \vdash \Delta; K : T$ and since evaluation context variables denote terms with a hole, this suggests to have:

$$\Delta \ ::= \ \epsilon \mid \Delta, \alpha : [U \vdash T]$$

Then, we need an axiom rule for conclusions:

$$\frac{(\alpha : [U \vdash T]) \in \Delta}{\Gamma; U \vdash \Delta; \alpha : T}$$

We are then ready for giving the following dual rules:

$$\frac{\Gamma, x : I \vdash M : I}{\Gamma \vdash \mathtt{cofix}_x.M : I} \qquad \frac{\Gamma; I \vdash \alpha : [I \vdash U]; K : U}{\Gamma; I \vdash \mathtt{fix}_\alpha.K : U}$$

(note that the symmetry would be perfect if in $LJ^T_{\mu\tilde{\mu}}$ instead of LJ_T)

Adding fixpoints and cofixpoints with parameters

Dependencies introduce a reading of the sequent from left to right. Let us consider the extended syntax:

$$\begin{array}{lcl} M, N, T, U & ++= & \mathtt{cofix}_x(\vec{y}).M \\ K & ++= & \alpha \mid \mathtt{fix}_\alpha(\vec{y}).K \end{array}$$

For cofixpoints, the rule scales easily using declarations of contexts:

$$\frac{\Gamma, x : [\overrightarrow{y : T} \vdash I\overrightarrow{N}], \overrightarrow{y : T} \vdash M : I\overrightarrow{N}}{\Gamma \vdash \mathtt{cofix}_x(\vec{y}).M : [\overrightarrow{y : T} \vdash I\overrightarrow{N}]}$$

For fixpoints (and we are still restricting ourselves to the tail-recursive case and no dependency in the conclusion), we need to type evaluation context variables with contexts too:

$$\Delta \ ::= \ \epsilon \mid \Delta, \alpha : [\Gamma; U \vdash T]$$

Then, the new rule is:

$$\frac{\Gamma, \overrightarrow{y : T}; I\overrightarrow{N} \vdash \alpha : [\overrightarrow{y : T}; I\overrightarrow{N} \vdash U]; K : U}{\Gamma; [\overrightarrow{y : T}; I\overrightarrow{N} \vdash U] \vdash \mathtt{fix}_\alpha(\vec{y}).K : U}$$

Adding fixpoints and cofixpoints with parameters: reduction rules

The reduction rules extend easily:

$$
\begin{array}{llcll}
M & (\mathtt{fix}_\alpha(\vec{y}).K)\overrightarrow{N} & \rightarrow & M & K\{\overrightarrow{N}/\vec{y}\}\{\mathtt{fix}_\alpha(\vec{y}).K/\alpha\} \\
(\mathtt{cofix}_x(\vec{y}).M)\overrightarrow{N} & K & \rightarrow & M\{\overrightarrow{N}/\vec{y}\}\{\mathtt{cofix}_x(\vec{y}).M/x\} & K
\end{array}
$$

Adding fixpoints and cofixpoints: the general case

In the non-tail recursive case, as e.g. in $f := \mathtt{fix}_f\, \lambda n.\mathtt{match}\, n\, \mathtt{with}\, 0 \rightarrow 0 \mid S\, n \rightarrow S\,(fn)$ **end**, we need to pass a continuation to the recursive evaluation-context variable. But in LJ_T a continuation is itself represented by an evaluation-context variable. Hence, we have a dependency of the recursive evaluation-context variable into another evaluation-context variable. This leads to the following generalised syntax:

$$
\begin{array}{lcl}
M, N, T, U & {+}{+}{=} & \mathtt{cofix}_x(\vec{y})M \\
K & {+}{+}{=} & \alpha \mid \mathtt{fix}_\alpha(\vec{y}\alpha)K \\
\Delta & ::= & \epsilon \mid \Delta, \alpha : [\Gamma; U \vdash \Delta; T]
\end{array}
$$

The axiom rule for conclusions does not change much:

$$
\frac{(\alpha : [\Gamma'; U \vdash \Delta'; T]) \in \Delta}{\Gamma; [\Gamma'; U \vdash \Delta'] \vdash \Delta; \alpha : T}
$$

Adding fixpoints and cofixpoints: the general case

Now, we need to build substitutions referring to evaluation contexts:

$$\frac{}{\Gamma \vdash \epsilon : [;V \vdash] \mapsto [;V \vdash]} \qquad \frac{\Gamma \vdash M_0 : U_0 \qquad \Gamma \vdash \overrightarrow{M}\overrightarrow{K} : [\Gamma'; V \vdash]\{M_0/x_0\} \mapsto [;V' \vdash]}{\Gamma \vdash M_0\overrightarrow{M}\overrightarrow{K} : [x_0 : U_0, \Gamma'; V \vdash] \mapsto [;V' \vdash]}$$

$$\frac{\Gamma; V_0 \vdash K : T_0 \qquad \Gamma \vdash \overrightarrow{K} : [\Gamma'; V \vdash \Delta;] \mapsto [;V' \vdash]}{\Gamma \vdash K_0\overrightarrow{K} : [\Gamma'; V \vdash \alpha : [V_0 \vdash T_0], \Delta;] \mapsto [;V' \vdash]}$$

And we need to apply these substitutions:

$$\frac{\Gamma; [\Gamma'; V \vdash \Delta; T] \vdash K' : T' \qquad \Gamma \vdash \overrightarrow{M}\overrightarrow{K} : [\Gamma'; V \vdash \Delta] \mapsto [;V' \vdash]}{\Gamma; V' \vdash K'\overrightarrow{M}\overrightarrow{K} : T'}$$

We are now ready to give the general rule for fixpoints:

$$\frac{\Gamma, \overrightarrow{y : T}; x : I\overrightarrow{N} \vdash \beta : [;P(\vec{y}, x) \vdash U], \alpha : [\overrightarrow{y : T}; x : I\overrightarrow{N} \vdash \beta : [;P(\vec{y}, x) \vdash U]; U]; K : U}{\Gamma; [\overrightarrow{y : T}; x : I\overrightarrow{N} \vdash \beta : [;P(\vec{y}, x) \vdash U]] \vdash \mathtt{fix}_\alpha(\vec{y}\beta).K : U}$$

(this complexity is the price to pay for tail-recursive simulation of non tail-recursive fixpoints)

Adding fixpoints and cofixpoints: the general case

The reduction rules does not change much:

$$M \;\; (\mathtt{fix}_\alpha(\vec{y}\beta).K)\overrightarrow{N}K' \quad \rightarrow \quad M \;\; K\{\overrightarrow{N}/\vec{y}\}\{K'/\beta\}\{\mathtt{fix}_\alpha(\vec{y}).K/\alpha\}$$

The non tail-recursive example is expressed like this:

$$\lambda n.\, n\, \mathtt{fix}_\alpha(\beta).[0 \rightarrow 0 \mid S\, n \rightarrow n(\alpha(\tilde{\mu}x.(S\, x)))]$$

Note that for building non linear evaluation contexts, we a priori need the following extra rule adapted from $LJ^T_{\mu\tilde{\mu}}$ to LJ_T:

$$\frac{\Gamma, x : A \vdash M : B}{\Gamma; A \vdash \tilde{\mu}x.M : B}$$

Symmetry of the guard conditions

We have the following symmetry:

Guard condition for fixpoint = recursion traverses at least one left introduction rule

Guard condition for cofixpoint = recursion traverses at least one right introduction rule

For inductive types and fixpoints, termination comes from the interaction between a finite term and an infinite guarded evaluation context.

For coinductive types and cofixpoints, termination comes from the interaction between a guarded infinite term and a finite evaluation context.

Note that in this duality, the difference between inductive and coinductive types is not a built-from-constructor vs built-from-destructors duality but a finite-infinite vs infinite-finite duality.