Dependent Type Theory in Polarised Sequent Calculus

Étienne Miquey\textsuperscript{1}, Xavier Montillet\textsuperscript{2}, and Guillaume Munch-Maccagnoni\textsuperscript{2}

\textsuperscript{1} CNRS, LSV, ÉNS Paris-Saclay, Inria, Cachan, France
\textsuperscript{2} Inria, LS2N CNRS, Nantes, France

Thanks to several works on classical logic in proof theory, it is now well-established that continuation-passing style (CPS) translations in call by name and call by value correspond to different polarisations of formulae (Girard, 1991; Danos, Joinet, and Schellinx, 1997; Laurent, 2002). Extending this observation and building on Curien and Herbelin’s abstract-machine-like calculi (2000), the last author proposed a term assignment for a polarised sequent calculus (where the polarities of formulae determine the evaluation order) in which various calculi from the literature can be obtained with macros responsible for the choices of polarities (Munch-Maccagnoni, 2013). It aims to explain several CPS translations from the literature by decomposing them through a single CPS for sequent calculus. It has later proved to be a fruitful setting to study the addition of effects and resource modalities (Curien, Fiore, and Munch-Maccagnoni, 2016), providing a categorical proof theory of Call By Push Value semantics (Levy, 2004).

We propose to bring together a dependently-typed theory (ECC) and polarised sequent calculus, by presenting a calculus L\textsubscript{dep} suitable as a vehicle for compilation and representation of effectful computations. As a first step in that direction, we show that L\textsubscript{dep} advantageously factorize a dependently typed continuation-passing style translation for ECC+call/cc. To avoid the inconsistency of type theory with control operators, we restrict their interaction. Nonetheless, in the pure case, we obtain an unrestricted translation from ECC to itself, thus opening the door to the definition of dependently typed compilation transformations.

Overview of L\textsubscript{dep} Recall that the key notion of term assignments for sequent calculi is that of a command, written \((t \parallel e)\), which can be understood as a state of an abstract machine, representing the evaluation of an proof (or expression) \(t\) against a counter-proof \(e\) that we call context. Their typing judgements are of the form \(\Gamma \vdash t : A \mid \Delta\) and \(\Gamma \vdash e : A \mid \Delta\), which correspond respectively to underlying sequents \(\Gamma \vdash A, \Delta\) and \(\Gamma, A \vdash \Delta\), in which \(A\) is in both cases the principal formula of the sequent. The command \((t \parallel e)\) is the result of applying the cut rule with \(t\) and \(e\) as premises: \(\langle t \parallel e \rangle : (\Gamma \vdash \Delta)\). It represents a cut rule with no principal formula.

But, in comparison to other presentations of sequent calculi, and like in Girard’s original formulation of LC, our logic features a negation operator \(\perp\) which is involutive strictly: \(A = A \perp\perp\). This involution allows us to represent any sequent \(c : (\Gamma \vdash \Delta)\) (resp. \(\Gamma \vdash t : A \mid \Delta\)) as a sequent \(c : (\perp \vdash A \mid \Delta)\) (resp. \(\perp \vdash \Gamma \perp \perp \mid t : A\)) with all formulae on the right. Thus, we are able to use a single grammar to describe both expressions and contexts.

The sequent calculus we propose is, in term of expressiveness, an extension of Luo’s ECC. Namely, ECC contains dependent products \(\Pi(x : A), B\) (becoming here a dependent \(\otimes\)), and dependent sums \(\Sigma(x : A), B\) (becoming here a dependent \(\circ\)), a cumulative hierarchy of universes \(\Box\), and an impredicative propositional universe \(P\), the inductive type of booleans with dependent elimination \(\text{B}\), and equalities between terms \(t = u:\)

<table>
<thead>
<tr>
<th>Atoms</th>
<th>(C := x \mid \Box \mid P \mid \Box \mid t = u)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Types\textsuperscript{2}</td>
<td>(P := C \mid A \otimes x.B \mid \downarrow A)</td>
</tr>
<tr>
<td>Types\textsuperscript{2}</td>
<td>(N := C \perp \mid A \otimes x.B \mid \hat{\uparrow} A)</td>
</tr>
<tr>
<td>Types</td>
<td>(A := P \mid N)</td>
</tr>
<tr>
<td>Values</td>
<td>(V := x \mid A \mid V \otimes A V' \mid \text{true} \mid \text{false} \mid \text{refl})</td>
</tr>
<tr>
<td>Terms</td>
<td>(t := \mu^x.x.c \mid \cdot \mid V^\circ)</td>
</tr>
<tr>
<td>Commands</td>
<td>(c := \langle t \parallel V \rangle^+)</td>
</tr>
</tbody>
</table>

where the notations \(\mu^x.x.c / \mu^x.x.c\) distinguish the binder according to the polarity of the corresponding type.
Since sequent calculi allow us to manipulate classical logic, we need to restrict dependencies to avoid logical inconsistencies (Herbelin, 2005). Following previous works (Herbelin, 2012; Miquey, 2019), we only allow negative-elimination-free (NEF) terms within types, which are thunkable (value-like) terms. In fact, we relax this constraint into that of Girard’s stoup (Girard, 1991), which similarly implies thunkability/linearity (Munch-Maccagnoni, 2013, IV.6). We take advantage of delimited control operators (in the form of \( \mu c \) and \( \cdot \)) to separate regular and dependent typing modes:

\[
\frac{\Gamma \vdash t : P \quad \Gamma \vdash v : P^\perp}{\Gamma \vdash \{t \parallel v\}^+ : (\Gamma^+) \quad \Gamma \in \text{NEF}}
\]

\[
\frac{c : (\Gamma, x : N) \quad \Gamma \vdash c : \mu^\circ x.c : N}{\Gamma \vdash \mu c : N} \quad \frac{c : (B \Gamma, x : N) \quad \Gamma \vdash c : \mu^\circ x.c : N}{\Gamma \vdash \mu c : N} \quad \mathbf{\cdot} \not\in B \quad \frac{\Gamma \vdash B \Gamma \vdash \cdot : B^\perp}{\Gamma \vdash B \Gamma \vdash \cdot : B^\perp}
\]

**Regular mode**

Observe that in the latter, the turnstile is annotated with a return type whose dependencies evolve with the typing derivation (see Miquey 2019 for more details). For instance, considering the type:

\[
\Gamma \vdash t : \Pi(X : P).X \cdot \cdot T(\text{true})
\]

we can inhabit it with the following term:

\[
\frac{\Gamma \vdash B \Gamma \vdash \cdot : B^\perp}{\Gamma \vdash B \Gamma \vdash \cdot : B^\perp}
\]

**Dependent mode**

Following the approach advocated in Boulier, Pédrot, and Tabareau (2017), the soundness of our system is proved by means of a syntactic model. In other words, we define a typed translation from our system to (an extension of) Luo’s ECC (1990). In broad lines, this translation follows the structure of the call-by-value continuation-passing style translation highlighted in Miquey (2019): we use dependent and parametric return types for continuations, and we translate NEF terms \( t \) at two different levels \( [t]_0 \) and \( [t]_1 \) in a way that is reminiscent of parametricity translations. For instance, the translations of a (closed and NEF) \( b \) boolean verify:

\[
[b]_1 : \Pi(R : \mathbb{B} \rightarrow \mathbb{P}).((\Pi(x : \mathbb{B}).R x) \rightarrow R [b]_0))
\]

Observe that by parametricity, this implies in particular that for any continuation \( k \) of parametric return type \( R \), we have \([b]_1 R k \equiv k [b]_0\), emphasizing that such a translation is only compatible with NEF terms that observationally behave like values.

Insofar as we can easily embed ECC+call/cc (evaluated in call by value) in our system, this translation allows us to factorize a CPS translation from this calculus to the (pure) ECC:

\[
\text{ECC + call/cc} \xrightarrow{\text{macros}} \text{I}_{\text{dep}} \xrightarrow{\text{CPS}} \text{ECC}
\]

Interestingly, by considering only the pure (by-value) ECC, we can define a dependently typed translation to itself without any kind of restriction on dependent types. Our translation improves over Bowman, Cong, Rioux, and Ahmed (2017) in that no extra assumption (in particular, we do not require an extensional type theory) are necessary to prove its soundness.

1A Coq development formalizing some aspects of these ideas is available at: https://www.irif.fr/~emiquey/content/CPS_ECC.v
References


