Toward dependent choice: a classical sequent calculus with dependent types

Hugo Herbelin, Étienne Miquey

Toward dependent choice : a classical sequent calculus with dependent types

Hugo Herbelin[1] and Étienne Miquey[1,2]

[1] PiR2, INRIA, Laboratoire PPS, Université Paris-Diderot
hugo.herbelin@inria.fr, emiquey@pps.univ-paris-diderot.fr
[2] IMERL, Universidad de la República, Montevideo

The dependent sum type of Martin-Löf's type theory provides a strong existential elimination, which allows to prove the full axiom of choice. The proof is simple and constructive:

$$\begin{array}{rcl} AC_A & := & \lambda H.(\lambda x.\,\mathtt{wit}(Hx), \lambda x.\,\mathtt{prf}(Hx)) \\ & : & \forall x^A \exists y^B P(x,y) \to \exists f^{A\to B} \forall x^A P(x, f(x)) \end{array}$$

where `wit` and `prf` are the first and second projections of a strong existential quantifier.

We present here a continuation of Herbelin's works [6], who proposed a way of scaling up Martin-Löf proof to classical logic. The first idea is to restrict the dependent sum type to a fragment of our system we call *N-elimination-free*, making it computationally compatible with classical logic. The second idea is to represent a countable universal quantification as an infinite conjunction. This allows to internalize into a formal system (called dPA^ω) the realizability approach [2, 5] as a direct proof-as-programs interpretation.

Informally, let us imagine that given $H : \forall x^A \exists y^B P(x,y)$, we have the ability of creating an infinite term $H_\infty = (H0, H1, \ldots, Hn, \ldots)$ and select its n^{th}-element with some function `nth`. Then one might wish that

$$\lambda H.(\lambda n.\,\mathtt{wit}(\mathtt{nth}\ n\ H_\infty), \lambda n.\,\mathtt{prf}(\mathtt{nth}\ n\ H_\infty))$$

could stand for a proof for $AC_{\mathbb{N}}$. However, even if we were effectively able to build such a term, H_∞ might contain some classical proof. Therefore two copies of H_n might end up being different according to their context in which they are executed, and then return two different witnesses. This problem could be fixed by using a shared version of H_∞, say

$$\lambda H.\,\mathtt{let}\ a = H_\infty\ \mathtt{in}\ (\lambda n.\,\mathtt{wit}(\mathtt{nth}\ n\ a), \lambda n.\,\mathtt{prf}(\mathtt{nth}\ n\ a)\,.$$

It only remains to formalize the intuition of H_∞. We do this by a stream $\mathtt{cofix}^0_{fn}(Hn, f(S(n)))$ iterated on f with parameter n, starting with 0 :

$$AC_{\mathbb{N}} := \lambda H.\,\mathtt{let}\ a = \mathtt{cofix}^0_{fn}\ (Hn, f(S(n))\ \mathtt{in}(\lambda n.\,\mathtt{wit}(\mathtt{nth}\ n\ a), \lambda n.\,\mathtt{prf}(\mathtt{nth}\ n\ a)\,.$$

Whereas the stream is, at level of formulæ, an inhabitant of a coinductive defined infinite conjunction $\nu^0_{Xn}(\exists P(0,y) \wedge X(n+1))$, we cannot afford to pre-evaluate each of its components, and then have to use a *lazy* call-by-value evaluation discipline. However, it still might be responsible for some non-terminating reductions. Our approach to prove a normalization property would be to interpret it in HA^ω through a negative translation. However, the sharing forces us to have a state-passing-style translation, whose small-step behaviour is quite far from the sharing strategy we have in natural deduction.

In a recent paper, Ariola *et al.* presented a way to construct a CPS-translation for a call-by-need version of the $\bar{\lambda}\mu\tilde{\mu}$-calculus [1], which allows some sharing facilities. Yet, this translation does not enjoy any typing property, and then does not give us a way of proving normalization. Moreover, the

$\bar{\lambda}\mu\tilde{\mu}$-calculus is typed with sequent calculus [4], which does not allow to manipulate dependent types immediately.

We propose to deal with both problems while proving the normalization of our system in two steps. First, we translate our calculus to an adequate version of the $\bar{\lambda}\mu\tilde{\mu}$-calculus that allows to manipulate dependent types on the N-elimination-free fragment. Then we will try to adapt the CPS-translation for call-by-need to our case, while adding it a type.

This work is currently in progress. For now, we managed to tackle the first problem, that is to construct a sequent calculus version of the initial language dPA^ω, During this talk, we intend to focus on this point, which turns out to be tricky, mainly because of the desynchronization of the dependency at the level of type in call-by-value. Let us look at the β-rule to get an insight of what happens. If we define the $\to_L$ and $\to_R$ rule as expected (where $A^{\perp\!\!\!\perp}$ is the type of a refutation of A):

$$\frac{\Gamma, a : A \vdash p : B}{\Gamma \vdash \lambda a.p : [a : A] \to B}\ \to_L \qquad \frac{\Gamma \vdash q : A \quad \Gamma \vdash e : B[q/a]^{\perp\!\!\!\perp} \quad q \notin \mathrm{Nef} \to a \notin A}{\Gamma \vdash q \cdot e : ([a : A] \to B)^{\perp\!\!\!\perp}}\ \to_R$$

and consider such a proof $\lambda a.p : [a : A] \to B$ and a context $q \cdot e : [a : A] \to B$, it reduces as follows :

$$\langle \lambda a.p \mid q \cdot e \rangle \rightsquigarrow \langle q \mid \tilde{\mu} a.\langle p \mid e \rangle \rangle$$

On the right side, we see that p, whose type is $B[a]$, is now cut with e of type $B[q]$. The idea is that in the full command a has been linked to q at a previous level of the typing judgement. We fixed this problem by making explicit a dependency list in the typing rules, which allows this typing derivation :

$$\dfrac{\Gamma \vdash q : A \qquad \dfrac{\dfrac{\Gamma, a : A \vdash p : B[a] \quad \Gamma, a : A \vdash e : B[q]; \{a|q\}}{\langle p \mid e \rangle : \Gamma, a : A; \{a|q\}}\ \text{CUT}}{\Gamma \vdash \tilde{\mu} a.\langle p \mid e \rangle : A^{\perp\!\!\!\perp}; \{.|q\}}\ \tilde{\mu}}{\langle q \mid \tilde{\mu} a.\langle p \mid e \rangle \rangle : \Gamma; \{\cdot|\cdot\}}\ \text{CUT}$$

By using this dependency list, we managed to fully translate the dPA^ω of [6] into a sequent calculus framework, that is a $\bar{\lambda}\mu\tilde{\mu}$-calculus with treatment of induction, cofix and equality. The translation is fully correct with respect to types.

The resulting calculus is given with a head-reduction, following a call-by-need evaluation strategy, and makes explicit the shared environment. This makes it a lot more closer to a small-step abstract machine than the original calculus, and it is our hope that as in [1] this would make the construction of a correct CPS-translation easier.

References

[1] Zena M. Ariola, Paul Downen, Hugo Herbelin, Keiko Nakata, and Alexis Saurin, *Classical call-by-need sequent calculi: The unity of semantic artifacts*, FLOPS 2012, Proceedings, 2012, pp. 32–46.

[2] Stefano Berardi, Marc Bezem, and Thierry Coquand, *On the computational content of the axiom of choice*, J. Symb. Log. **63** (1998), no. 2, 600–622.

[3] Ulrich Berger and Paulo Oliva, *Modified bar recursion*, Mathematical Structures in Computer Science **16** (2006), no. 2, 163–183.

[4] Pierre-Louis Curien and Hugo Herbelin, *The duality of computation*, ICFP, 2000, pp. 233–243.

[5] Martín H. Escardó and Paulo Oliva, *Bar recursion and products of selection functions*, CoRR **abs/1407.7046** (2014).

[6] Hugo Herbelin, *A constructive proof of dependent choice, compatible with classical logic*, Logic in Computer Science, LICS 2012, Proceedings, IEEE Computer Society, 2012, pp. 365–374.