Toward a computational reduction of dependent choice in classical logic to system F

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

[1] PiR2, INRIA, Institut de Recherche en Informatique Fondamentale, 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:

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

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

We present here a proof system which provides a proof-as-program interpretation of classical arithmetic with dependent choice, together with a computational reduction of this calculus to an intuitionistic one by means of a continuation-and-state-passing style translation. This system is a sequent-calculus version of Herbelin's dPA^ω calculus [5], who proposed a way of scaling up Martin-Löf proof to classical logic. The main ideas are first to restrict the dependent sum type to a fragment of the calculus to make it computationally compatible with classical logic, second to represent a countable universal quantification as an infinite conjunction. This allows to internalize into a formal system the realizability approach [2, 4] 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 :

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

Whereas the stream is, at level of formulæ, an inhabitant of a coinductively 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 thus have to use a *lazy* call-by-value evaluation discipline. However, it still might be responsible for some non-terminating reductions.

We intend to tackle the problem by progressively reducing the consistency of our system to the normalization of Girard-Reynold's system F. However, the sharing forces us to design a state-passing style translation, whose small-step behaviour is quite far from the sharing strategy in natural deduction. Besides, in order to get a proof of normalization through such a translation, we also need to guarantee some typing properties in the source language and along the translation.

We presented a preliminary version of this work at TYPES 2015, where, as a first step, we managed to develop a sequent-calculus version of dPA^ω, adapting the call-by-need version of the $\bar{\lambda}\mu\tilde{\mu}$-calculus designed by Ariola et al. [1]. Incidentally, we had to ensure its compatibility with dependent types, since the $\bar{\lambda}\mu\tilde{\mu}$-calculus [3] does not allow it directly. This led us to a type system annotated with a dependencies list, and made us add delimited continuations to our language. Indeed, if we consider the case of a proof $\lambda a.p : [a : A] \to B$ cut with a context $q \cdot e$ where $q : A$ and $e : B[q]^{\perp\!\!\perp}$, it usually reduces to the command $\langle q \mid \tilde{\mu}a.\langle p \mid e\rangle\rangle$ where $p : B[a]$ and $e : B[q]^{\perp\!\!\perp}$ are of incompatible types. While a annotation (to link a and q) on the type system can solve this, there is no hope that a direct continuation-passing style translation could be well-typed. Thus we introduced delimited continuations to turn it into a command $\langle \mu\hat{\mathfrak{tp}}.\langle q \mid \tilde{\mu}a.\langle p \mid \hat{\mathfrak{tp}}\rangle\rangle \mid e\rangle$ where p will not be cut with e until a is replaced by q.

The work is still in progress and in this talk, we propose to focus on the second step, that is the design of a continuation-and-state-passing style translation that is correct with respect to types and computation. As in [1], we benefited from Danvy's methodology of semantic artifacts. We first derive a small-step reduction system, to obtain a context-free abstract machine in which at each step a decision over a command $\langle p \mid e\rangle$ can be made by examining either the proof p or the context e in isolation. To do so, we separate the reductions rule in two different layers, which intuitively correspond to the call-by-value and store-management for the first one, and to the core computations for the second one.

This small-step system almost gives us directly a state-passing style translation. The remaining difficulty is to type the store in the target language, which is a quite subtle problem due to the fact that the store can be expanded in a non-linear way when unfolding a `cofix`. It is our hope that we could use the second-order quantification of system F to encode the store and its expansion, which would provide us with a proof of equiconsistency between classical arithmetic with dependent choice and system F.

Surprisingly, it turns out that our construction does not require any use of dependent choice at the meta-level. If some previous works [2, 6] succeeded in giving a computational content to the axioms of dependent choice or bar induction, this is to the best of our knowledge the first one that does not need any meta-use of one of these axioms.

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

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

[5] 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.

[6] J.-L. Krivine, *Dependent choice, 'quote' and the clock*, Th. Comp. Sc. **308** (2003), 259–276.