How to define dependently typed CPS using delimited continuations

Étienne Miquey[1]

Équipe Gallinette, INRIA
LS2N, Université de Nantes
etienne.miquey@inria.fr

There are a handful of reasons to pay attention to various flavors of dependently typed translations for dependently typed calculi. For instance, they may allow to ensure the preservation of safeness properties (obtained via dependent types) through a compilation process. Or, as defended by Boulier *et al.* [3], such translations can allow us to define syntactical models for type theories extended with new reasoning principles. In particular, a reasonable plan of attack to allow effectful programs in Coq could be to build successive layers extending CIC with side effects and to justify their soundness by means of syntactic translations. This path has been undertaken recently by Pédrot and Tabareau in [7, 8] to add various classes of effects to CIC. In that perspective, *continuation-passing style* translations give a semantics to control operators, that is to say classical logic, by expliciting the flow of control. Even though it is well-known that classical logic and dependent types can only be mixed if their interactions are restricted, it might still be of great interest to allow more classical reasoning in proof assistants by means of control operators.

In 2002 [2], Barthe and Uustalu first stated the impossibility of CPS translating dependent types. As observed in Bowman *et al.* [4], this result is due to the standard definition of typed CPS translation by double negation. They indeed manage to circumvent this point by using parametric *answer-types* in the translation at the price of an ad-hoc application constructor and of considering an extensional type theory as target language. During this talk, we intend to present a more general method that we introduced to CPS translate a call-by-value sequent calculus with dependent types [5]. Our construction relies on the use of *delimited continuations* in the source language, leading to more parametric answer-types in the translation. The latter turn out to be enough to soundly type the CPS without further addition. We shall now briefly outline the rationale guiding our use of delimited continuations with that respect [5].

It is folklore that sequent calculi are in essence close to the operational semantics of abstract machines, which makes them particularly suitable to define CPS translations. We take advantage of their fine-grained reduction rules to observe the problem already in the source language that we defined as a call-by-value dependently typed calculus. Having a look at the β-reduction rule gives us an insight of what happens. Informally, consider a dependent function $\lambda a.p : \Pi a : A.B$ (*i.e.* p is of type $B[a]$) that is executed in front of a stack $q \cdot e : \Pi a : A.B$ (*i.e.* e is of type $B[q]$). A call-by-value head-reduction rule (in a $\lambda\mu\tilde{\mu}$-like fashion) for this command would then produce a command that we cannot type:

$$\langle \lambda a.p | q \cdot e \rangle \rightsquigarrow \langle q | \tilde{\mu} a.\langle p | e \rangle \rangle \qquad \dfrac{\dfrac{\Pi_q}{\Gamma \vdash q : A \mid \Delta} \quad \dfrac{\dfrac{\Gamma, a : A \vdash p : \cancel{B[a]} \mid \Delta \quad \Gamma, a : A \mid e : \cancel{B[q]} \vdash \Delta}{\langle p | e \rangle : \Gamma, a : A \vdash \Delta}\ \textit{Mismatch}}{\Gamma \mid \tilde{\mu} a.\langle p | e \rangle : A \vdash \Delta}\ (\tilde{\mu})}{\langle q | \tilde{\mu} a.\langle p | e \rangle \rangle : \Gamma \vdash \Delta}\ (\textsc{Cut})$$

The intuition is that in the full command, a has been linked to q at a previous level of the typing judgment. However, the command is still computationally safe, in the sense that the head-reduction imposes that the command $\langle p | e \rangle$ will not be executed before the substitution of a by q is performed. By then, the problem would be solved. This phenomenon can be seen as a desynchronization of the typing process with respect to computation.

Interestingly, the very same happens when trying to define a CPS translation carrying type dependencies. Indeed, a translation of the command above is very likely to look like:

$$[\![q]\!]\,[\![\tilde{\mu}a.\langle p|e\rangle]\!] = [\![q]\!]\,(\lambda a.([\![p]\!]\,[\![e]\!])),$$

where $[\![p]\!]$ is intuitively of type $\neg\neg B[a]$ and $[\![e]\!]$ of type $\neg B[q]$, hence the sub-term $[\![p]\!]\,[\![e]\!]$ is ill-typed.

We follow the idea that the correctness should be guaranteed by a head-reduction strategy, preventing $\langle p|e\rangle$ from reducing before the substitution of a was made. We would like to ensure the same property in the target language, namely that $[\![p]\!]$ cannot be applied to $[\![e]\!]$ before $[\![q]\!]$ has furnished a value to substitute for a. Assuming that q eventually produces a value V, we are informally looking for the following translation and the corresponding reduction sequence:

$$[\![q]\!]\,[\![\tilde{\mu}a.\langle p|e\rangle]\!] \overset{?}{=} ([\![q]\!](\lambda a.[\![p]\!]))[\![e]\!] \to ((\lambda a.[\![p]\!])\,[\![V]\!])\,[\![e]\!] \to [\![p]\!][[\![V]\!]/a]\,[\![e]\!]$$

Since $[\![p]\!][[\![V]\!]/a]$ has a type convertible to $\neg\neg B[q]$, the last term is now well-typed.

The first observation is that the term $([\![q]\!](\lambda a.[\![p]\!]))[\![e]\!]$ could be typed by turning the type $A \to \bot$ of the continuation that $[\![q]\!]$ is waiting for into a (dependent) type $\Pi a : A.R[a]$ parameterized by R. This way we could have $[\![q]\!] : \forall R.(\Pi a : A.R[a] \to R[q])$ instead of $[\![q]\!] : ((A \to \bot) \to \bot)$. For $R[a] := (B(a) \to \bot) \to \bot$, the whole term is well-typed. Readers familiar with realizability will also note that such a term is realizable, since it eventually terminates on a correct term $[\![p[q/a]]\!]\,[\![e]\!]$.

The second observation is that such a term suggests the use of delimited continuations [1] to temporarily encapsulate the evaluation of q when reducing such a command. Indeed, the use of delimited continuations allows the source calculus to mimic the aforedescribed reduction:

$$\langle \lambda a.p|q \cdot e\rangle \rightsquigarrow \langle \mu\hat{\mathfrak{p}}.\langle q|\tilde{\mu}a.\langle p|\hat{\mathfrak{p}}\rangle\rangle|e\rangle \rightsquigarrow \langle \mu\hat{\mathfrak{p}}.\langle V|\tilde{\mu}a.\langle p|\hat{\mathfrak{p}}\rangle\rangle|e\rangle \rightsquigarrow \langle \mu\hat{\mathfrak{p}}.\langle p[V/a]|\hat{\mathfrak{p}}\rangle|e\rangle \rightsquigarrow \langle p[V/a]|e\rangle$$

Incidentally, this allows us to introduce a list of dependencies within the typing derivations of judgments involving delimited continuations, and to fully absorb the potential inconsistency in the type of $\hat{\mathfrak{p}}$.

Finally, we shall explain how the translation of dependent sums dually requires co-delimited continuations, how the use of delimited continuations also unveils the need for a restriction to safely use control operators, and how we plan to reuse this method to define a sequent calculus presentation of CIC or even an extension of Munch-Maccagnoni's polarised system L [6] to dependent types.

References

[1] Zena M. Ariola, Hugo Herbelin, and Amr Sabry, *A type-theoretic foundation of delimited continuations*, Higher-Order and Symbolic Computation **22** (2009), no. 3, 233–273.

[2] Gilles Barthe and Tarmo Uustalu, *CPS translating inductive and coinductive types*, Workshop on Partial Evaluation and Semantics-Based Program Manipulation (PEPM '02), Proceedings, ACM, 2002, pp. 131–142.

[3] Simon Boulier, Pierre-Marie Pédrot, and Nicolas Tabareau, *The next 700 syntactical models of type theory*, 6th Conference on Certified Programs and Proofs, CPP 2017, Proceedings, ACM, 2017, pp. 182–194.

[4] William J. Bowman, Youyou Cong, Nick Rioux, and Amal Ahmed, *Type-preserving CPS translation of* Σ *and* Π *types is not not possible*, Proc. ACM Program. Lang. **2** (2018), no. POPL, 22:1–22:33.

[5] Étienne Miquey, *A classical sequent calculus with dependent types*, 26th European Symposium on Programming, ESOP 2017, Proceedings (Hongseok Yang, ed.), LNCS, vol. 10201, Springer, 2017, pp. 777–803.

[6] Guillaume Munch-Maccagnoni, *Focalisation and classical realisability*, Computer Science Logic, 23rd international Workshop, CSL 2009, Proceedings, LNCS, vol. 5771, Springer, 2009, pp. 409–423.

[7] Pierre-Marie Pédrot and Nicolas Tabareau, *An effectful way to eliminate addiction to dependence*, 32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, IEEE Computer Society, 2017, pp. 1–12.

[8] ———, *Failure is Not an Option - An Exceptional Type Theory*, 27th European Symposium on Programming, ESOP 2018, Proceedings (Amal Ahmed, ed.), LNCS, vol. 10801, Springer, 2018, pp. 245–271.