The Münchhausen method and combinatory type theory

Thorsten Altenkirch[1], Ambrus Kaposi[2]*, Artjoms Šinkarovs[3]†, and Tamás Végh[2]

[1] School of Computer Science, University of Nottingham, UK
psztxa@nottingham.ac.uk
[2] Eötvös Loránd University, Budapest, Hungary
{akaposi,vetuaat}@inf.elte.hu
[3] Heriot-Watt University, Scotland, UK
a.sinkarovs@hw.ac.uk

Abstract

In one of his tall tales, Baron Münchhausen pulled himself out of a swamp by his own hair. Inspired by this, we present a technique to justify "very dependent types": terms with types that include the term itself. The Münchhausen method is an informal way to make this precise. We don't have to resort to untyped preterms or typing relations, the technique works in a completely algebraic setting (such as categories with families). We present the method through a series of examples.

A dependent version of "products as functions from Bool"

There is a well known type isomorphism

$$A \times B \;\cong\; (b : \mathsf{Bool}) \to \mathsf{if}\, b\, \mathsf{then}\, A\, \mathsf{else}\, B.$$

Can we turn the nondependent product into a Σ type? Given $A : \mathsf{Type}$, $B : A \to \mathsf{Type}$, we want something like

$$\Sigma\, A\, B \;\cong\; (b : \mathsf{Bool}) \to \mathsf{if}\, b\, \mathsf{then}\, A\, \mathsf{else}\, (B\, \square),$$

but we don't know what to put in the placeholder $\square$. It should be the output of the function when the input is $b = \mathsf{true}$. Once the function is given a name, we can refer to it:

$$f : (b : \mathsf{Bool}) \to \mathsf{if}\, b\, \mathsf{then}\, A\, \mathsf{else}\, (B\, (f\, \mathsf{true}))$$

This is sometimes called a "very dependent type" [3]: the term f appears in its own type. It is possible to make sense of such a type using preterms and typing relations [1], but we can also present it algebraically as follows.

$$\begin{aligned} &a_0 : A \\ &f \;\; : (b : \mathsf{Bool}) \to \mathsf{if}\, b\, \mathsf{then}\, A\, \mathsf{else}\, (B\, a_0) \\ &\not{a}_0 : a_0 = f\, \mathsf{true} \end{aligned}$$

We first declare a_0 as the extra component that the type of f will depend on. Then we can declare f itself with the help of a_0. Finally, knowing about f, we can equate a_0 away: after learning about $\not{a}_0$, we know that the type of f is $(b : \mathsf{Bool}) \to \mathsf{if}\, b\, \mathsf{then}\, A\, \mathsf{else}\, (B\, (f\, \mathsf{true}))$. We only needed a_0 for bootstrapping the type of f.

*Ambrus Kaposi was supported by the "Application Domain Specific Highly Reliable IT Solutions" project which has been implemented with support from the National Research, Development and Innovation Fund of Hungary, financed under the Thematic Excellence Programme TKP2020-NKA-06 (National Challenges Subprogramme) funding scheme, and by Bolyai Scholarship BO/00659/19/3.

†This work is supported by the Engineering and Physical Sciences Research Council through the grant EP/N028201/1.

Type theory without types

A well-known example of the same technique is equating the sort of types away. We declare the sorts and operations of type theory as follows.

$$\begin{aligned}
&\mathsf{Con} : \mathsf{Type}\\
&\mathsf{Ty} \;\; : \mathsf{Con} \to \mathsf{Type}\\
&\mathsf{Tm} \;\; : (\Gamma : \mathsf{Con}) \to \mathsf{Ty}\,\Gamma \to \mathsf{Type}\\
&\mathsf{U} \;\; : \mathsf{Ty}\,\Gamma\\
&\cancel{\mathsf{Ty}} \;\; : \mathsf{Ty}\,\Gamma = \mathsf{Tm}\,\Gamma\,\mathsf{U}
\end{aligned}$$

We have to introduce types, but once we have $\cancel{\mathsf{Ty}}$, we know that types are just terms of type U. Note that such a theory would be inconsistent through Russell's paradox, but it is easy to fix this by stratification (adding natural number indices to Ty and U, see [5]). Thus Münchhausen provides an algebraic way to define universes à la Russell, that is, a type theory without types: we use types only for bootstrapping.

Type theory without contexts

Just as we equated types away, we can do the same with contexts and substitutions. We start with the signature for categories with families (CwF [2]) with the four sorts $\mathsf{Con} : \mathsf{Type}$, $\mathsf{Sub} : \mathsf{Con} \to \mathsf{Con} \to \mathsf{Type}$, $\mathsf{Ty} : \mathsf{Con} \to \mathsf{Type}$, $\mathsf{Tm} : (\Gamma : \mathsf{Con}) \to \mathsf{Ty}\,\Gamma \to \mathsf{Type}$, empty context $\diamond : \mathsf{Con}$, context extension $-\triangleright- : (\Gamma : \mathsf{Con}) \to \mathsf{Ty}\,\Gamma \to \mathsf{Con}$, and we have $\top$ and Σ types. Then we add equations such as $\mathsf{Con} = \mathsf{Ty}\,\diamond$, $\mathsf{Sub}\,\Delta\,\Gamma = \mathsf{Tm}\,\Delta\,(\Gamma[\epsilon])$, $\sigma\circ\delta = \sigma[\delta]$, $\Gamma\triangleright A = \Sigma\,\Gamma\,(A[\mathsf{q}])$. In the end we have e.g. $\mathsf{Tm} : (\Gamma : \mathsf{Ty}\,\top) \to \mathsf{Ty}\,\Gamma \to \mathsf{Type}$ and $\Sigma : (A : \mathsf{Ty}\,\Gamma) \to \mathsf{Ty}\,(\Sigma\,\Gamma\,(A[\mathsf{q}])) \to \mathsf{Ty}\,\Gamma$.

The language with all these equations can be justified: any CwF with $\top$ and Σ can be turned into an equivalent CwF which satisfies all these equations. There is an analogous model construction for type theory with types, and the two model constructions can be combined.

Towards type theory without contexts for real

Simply typed combinator calculus is a language where contexts are not even present for bootstrapping [4]. There are no variables, function space is built-in and the combinators S and K are used to define functions. Due to the technical challenges which come with not being able to use variables [6], combinator calculus was never extended to dependent types. We are in the process of defining a combinatory (dependent) type theory using Münchhausen's technique.

We first introduce types $\mathsf{Ty} : \mathsf{Type}$, terms indexed by types $\mathsf{Tm} : \mathsf{Ty} \to \mathsf{Type}$, a universe $\mathsf{U} : \mathsf{Ty}$ with the equation $\mathsf{Ty} = \mathsf{Tm}\,\mathsf{U}$, then we introduce families $- \Rightarrow\mathsf{U} : \mathsf{Ty} \to \mathsf{Ty}$ with instantiation $- \$ - : \mathsf{Tm}\,(A \Rightarrow\mathsf{U}) \to \mathsf{Tm}\,A \to \mathsf{Ty}$. Now we are in the position of declaring dependent function space $\Pi : (A : \mathsf{Ty}) \to \mathsf{Tm}\,((A \Rightarrow\mathsf{U}) \Rightarrow\mathsf{U})$ and application $- \cdot - : \mathsf{Tm}\,(\Pi\,A\,\$\,B) \to (a : \mathsf{Tm}\,A) \to \mathsf{Tm}\,(B\,\$\,a)$. We introduce constant families $\mathsf{K_f} : \mathsf{Ty} \to \mathsf{Tm}\,(B \Rightarrow\mathsf{U})$ with the equation $\mathsf{K_f}\,A\,\$\,b = A$ which allows us to express the non-dependent function type $A \Rightarrow B := \Pi\,A\,\$\,(\mathsf{K_f}\,B)$. Using a helper combinator $- \Rightarrow_\mathsf{K} - : \mathsf{Tm}\,(A \Rightarrow\mathsf{U}) \to \mathsf{Ty} \to \mathsf{Tm}\,(A \Rightarrow\mathsf{U})$ with equation $(B \Rightarrow_\mathsf{K} C)\,\$\,a = B\,\$\,a \Rightarrow C$, we can can express the dependent version of the K combinator $\mathsf{K} : \mathsf{Tm}\,(\Pi\,A\,\$\,B \Rightarrow_\mathsf{K} A)$ with equation $\mathsf{K}\cdot a\cdot b = a$. We can do the same for the dependently typed S combinator.

Our goal is to show that the syntax of combinatory type theory is equivalent to the syntax of CwF with $\top$, Σ, Π and universes.

References

[1] Henk Barendregt. Introduction to generalized type systems. *J. Funct. Program.*, 1(2):125–154, 1991.

[2] Simon Castellan, Pierre Clairambault, and Peter Dybjer. Categories with families: Unityped, simply typed, and dependently typed. *CoRR*, abs/1904.00827, 2019.

[3] Jason J. Hickey. Formal objects in type theory using very dependent types. In *In Foundations of Object Oriented Languages 3*, 1996.

[4] J. Roger Hindley and Jonathan P. Seldin. *Introduction to Combinators and Lambda-Calculus*. Cambridge University Press, 1986.

[5] András Kovács. Generalized universe hierarchies and first-class universe levels. In Florin Manea and Alex Simpson, editors, *30th EACSL Annual Conference on Computer Science Logic, CSL 2022, February 14-19, 2022, Göttingen, Germany (Virtual Conference)*, volume 216 of *LIPIcs*, pages 28:1–28:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

[6] Conor McBride. What is the combinatory logic equivalent of intuitionistic type theory? Answer to question on StackOverflow, 2012. https://stackoverflow.com/questions/11406786/what-is-the-combinatory-logic-equivalent-of-intuitionistic-type-theory.