# The Long Tale of Implementing CIC over CoC

Technical Article

Paul Lyutko, Groupoid Infinity

Kyiv 2016

# Contents

# 1 Introduction: The Why

Ladies and gentlemen! Let us start from the beginning. In the beginning there were quantifiers and lambdas binding variables in the terms. This is what any Pure Type System is built from. It also can be seen as a category made from types (as objects) and functions (as morphisms). If we want we can extend this category with any constructions we need but do it really need to be extended? What if it already contains all kind of structures we know? Hereby we claim — yes, it contains. The structures we seek are inductive types given by their constructors and eliminators, and coinductive types, and higher inductive types, and also a plenty of types formed by appropriate adjoint functors.

Such approach allows us to implement the full-featured language (EXE compiler) with dependent types atop of a small logical core of pure lambda calculus (OM compiler). You know the LISP language uses "code as data" metaphor, but we use "data as code" insteed!

It this text we discuss an encoding. An encoding is a model of one type-theory-like formal system in another type-theory-like formal system. The encoding we use implements axioms of Calculus of Inductive Construction (CIC), the type theory of inductive types, in Calculus of Construction, one of Pure Type System (PTS). We do it mapping types not into types but into a kind of structure known as setoids providing a well-behaved equality on our types and on the whole category. Also we utilize the fact that this category have limits of functors going into it, a categorical notion refining dependent function type from PTS.

Attempts to explore encodings are not new. The historical example is the Church encoding for natural numbers. There are generalizations of it [Berarducci]. Nowadays there are pragmatically charged compilers using them [Gonzalez]. Insteed we introduce encoding capable of implement dependent eliminator a.k.a. induction principle build by refining Berarducci encoding with categorical limits. Attention: we do not extend the preexistent category of types, we just discover all required kinds of structures in the category of setoids. Here it goes. Enjoy!

# 6 References

## References


[1] Henk Barendregt *The Lambda Calculus. Its syntax and semantics* 1981

[2] Henk Barendregt *Lambda Calculus With Types* 2010

[3] Erik Meijer, Symon Peyton Jones *Henk: a typed intermediate language* 1984

[4] Per Martin-Lf *Intuitionistic Type Theory* 1984