Isomorphism of Finitary Inductive Types

Nicolai Kraus and Christian Sattler

University of Nottingham

Abstract

We present an algorithm for deciding isomorphism of finitary mutually inductive types.

We consider the simply typed lambda calculus with finite products and coproducts, type variables X, Y, Z, ..., and mutually inductive types, for example defined by

$$\begin{aligned} A &= Z + XA^2 + BC \\ B &= 1 + BC \\ C &= Y + AC, \end{aligned}$$

which we call (parameterized) *finitary inductive types*. Equivalently, we can define these as nested μ-expression; for example, C can be represented as

$$\mu C.Y + (\mu A.Z + XA^2 + (\mu B.1 + CB)C)C.^1$$

Note that the attribute *finitary* indicates the absence of function types.

Based on discussions with Swierstra and Morris, Altenkirch [1] notices that the special case of *regular* types, i.e. types defined mutually as sums over products where each product contains at most one recursive variable, corresponds to proof-relevant regular grammars where terminal symbols commute. [2] They discuss isomorphism of such types in the set model and conjectures that isomorphism of regular types is decidable, while isomorphism of general finitary inductive types might be undecidable.

In this talk, we present an algorithm that decides syntactic isomorphism of general finitary inductive types with respect to the $\beta\eta$-equational theory with strong sums and interpretation of μ-expressions as initial algebras. We further show that the set model is complete with respect to this question. This yields the solution to Altenkirch's conjectures as a special case.

We want to outline one core observation on which the easier set model variant of this algorithm is based. As is well known, in the set model parameterized finitary inductive types can equivalently be expresses as a formal power series, e.g. via the intermediate notion of finitary containers. For example, lists over X (that is $\mu A.1 + AX$) admit the representation

$$1 + X + X^2 + X^3 + X^4 + \ldots, \tag{1}$$

while the series of the type C defined above starts with (we list all summands which have less than four occurrences of X, Y, Z)

$$Y + Y^2 + YZ + 3Y^3 + 3Y^2Z + YZ^2 + \ldots. \tag{2}$$

It is possible that certain coefficients of the power series corresponding to a type are not finite; for example, the natural numbers $\mu A.1 + A$ have a power series over zero variables with a single

[1] We write XA^2 for $X \times A \times A$.

[2] Confusingly, other authors have introduced the term *regular functors* over type variables for the concept we name finitary inductive types here.

summand of degree 0 with coefficient of cardinality $\mathbb{N}$. This imposes additional difficulties. It forces us to treat the part consisting of the infinite coefficients separately, a finitary inductive description of which we call the *unguarded* part. However, the surgery necessary on a finitary inductive type to effectively renove the unguarded part is not canonical, leaving a residue type having only finite power series coefficients that is in general not uniquely determined.

In the set model, two types are isomorphic if the coefficients of their corresponding power series are equal. The key point now is that the guarded part is always *algebraic* over a certain function field, i.e. it is has a corresponding *minimal polynomial*, a finite representation, with coefficients in a certain kind of finitely describable algebraic structure, and two of these finite representations can be algorithmically compared. The guarded part is however selectively masked by the unguarded part, requiring further algebraic machinery involving modules, lattices, and known methods for constructively dealing with systems of polynomial equations.

Having established isomorphism in the set model between two finitary inductive types, actually synthesizing one such isomorphism as a syntactic λ-term (and thus establishing completeness of the set model) requires internalizations of parts of the above arguments into a theory not even able to reflect many basic inductive arguments. Thus, we want to stress that the actual main technical effort concerns this aspect. We strive to give our arguments in an abstract fashion, preferring type- or category-level reasoning wherever possible (e.g., making use of categorical properties of traversable functors [3]). Still, reducing the amount of low-level details in this part of our proof warrants further attention.

Our project is superficially related to previous work by Fiore [2]. Crucially, however, our recursive types are not generic, but initial: instead of just constructing and destructing elements of inductive types, we have the full power of unique existence of the folding eliminator at hand; this induces a much stronger equational theory that is more difficult to reason about.

References

[1] Thorsten Altenkirch. Isomorphisms on inductive types (talk), 2005.

[2] Marcelo Fiore. Isomorphisms of generic recursive polynomial types. In *Proceedings of the 31st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '04, pages 77–88, New York, NY, USA, 2004. ACM.

[3] Mauro Jaskelioff and Ondrej Rypacek. An investigation of the laws of traversals. In *MSFP*, pages 40–49, 2012.