Annals of Pure and Applied Logic 169 (2018) 637–655

Contents lists available at ScienceDirect

Annals of Pure and Applied Logic

www.elsevier.com/locate/apal

From realizability to induction via dependent intersection

Aaron Stump

Computer Science, MacLean Hall, The University of Iowa, Iowa City, IA 52242, United States

ARTICLE INFO

Article history:
Received 16 October 2016
Received in revised form 8 January 2018
Accepted 7 March 2018
Available online 13 March 2018

MSC:
03B15
03B40
68N18
68N30

Keywords:
Extrinsic typing
Lambda encodings
Derivable induction
Internalized realizability

ABSTRACT

In this paper, it is shown that induction is derivable in a type-assignment formulation of the second-order dependent type theory $\lambda P2$, extended with the implicit product type of Miquel, dependent intersection type of Kopylov, and a built-in equality type. The crucial idea is to use dependent intersections to internalize a result of Leivant's showing that Church-encoded data may be seen as realizing their own type correctness statements, under the Curry–Howard isomorphism.

1. Introduction

Constructive type theory has been proposed as a foundation for constructive mathematics, and has found numerous applications in Computer Science, thanks to the Curry–Howard correspondence between constructive logic and pure functional programming [29,25,13]. Pure Type Systems (PTSs) are one formalism for constructive type theory, based on pure lambda calculus [6]. PTSs have very compact syntax, reduction semantics, and typing rules, which is appealing from a foundational and metatheoretic perspective. Unfortunately, PTSs by themselves have not been found suitable as a true foundation for constructive type theory in practice, due to the lack of inductive types. At the introduction of the Calculus of Constructions (CC), an important impredicative PTS, induction was lacking [11]. This led the inventors of CC and their collaborators to extend the theory with a primitive notion of inductive types, resulting in the Calculus of Inductive Constructions (CIC), which is the core formalism of the prominent Coq computer-proof soft-

E-mail address: aaron-stump@uiowa.edu.

https://doi.org/10.1016/j.apal.2018.03.002
0168-0072/

ware [45,46,35,12]. In 2001, this skepticism about induction in PTSs was solidified when Geuvers proved that induction is not derivable in second-order dependent type theory ($\lambda P2$), a subsystem of CC [20].

The adoption, in CIC and Coq, of a system for declaring primitive datatypes solved the problem of induction, and hence allowed formalization of a variety of results in Mathematics and Computer Science. A notable example among these is a Coq proof of the Four Color Theorem, which, unlike the theorem's original computer proof, does not depend on unverified programs for checking a large of number of special cases [23,4]. Thanks to the Curry–Howard isomorphism, such programs can be written and, crucially, proved sound within the type theory. So the addition of primitive datatypes opened up the possibility of formalizing complex mathematical results in type theory, that was lacking in pure CC. Other type theories provide mechanisms for defining inductive types. In Automath, for example, inductive types are defined axiomatically, simply by writing down constructors and asserting that induction holds [14]. In Martin-Löf type theory, one can use W-types to define inductive types, thus avoiding the need to add axioms to the theory for each new type; rather, the theory is extended once, with a single set of axioms for W-types [30]. Nevertheless, in all these cases, the pure type-theoretic core must be extended with additional operations, at both term and type level, to represent inductive types. At the term level, this necessitates additions to the usual proof of confluence of reduction of terms. In all cases, the resulting theory has now additional machinery requiring nontrivial metatheoretic analysis.

In this paper, we present an extension of a type-assignment formulation of $\lambda P2$, in which induction is derivable, indeed in two slightly different ways. The extension does not in any direct way correspond simply to adding primitive inductive types or induction principles. Rather, the extension strengthens the expressiveness of the dependent typing of $\lambda P2$, to take advantage of the computational power that is already present in impredicative type theory. The extension is with three constructs, all somewhat exotic but none new. The first is the implicit product $\forall\, x : A.\, B$ of Miquel, which allows one to generalize x of type A without introducing a λ-abstraction at the term level [32]. Second is the dependent intersection type of Kopylov [26]. Intersection types have been studied for many years in theoretical Computer Science, due to their strong connection with normalization properties (see [7] for a magisterial presentation). If a term t can be assigned types A and B, then it can also be assigned the type $A \cap B$. With dependent intersections, this is strengthened to: if a term t can be assigned types A and $[t/x]B$ (the substitution of t for x in B), then it can also be assigned the type $x : A \cap B$. In this paper, we will use the prefix notation $\iota\, x : A.\, B$, instead of Kopylov's $x : A \cap B$. The third construct in the extension is a primitive equality type, allowing expression of equality between terms x and y both of some common type A. While all three constructs are necessary for the derivations given of induction, the dependent intersections are most central to the construction, and so we will denote the resulting system $\iota\lambda P2$.

For nontrivial intersection types to be inhabited, we must work in a Curry-style (sometimes also called *extrinsic*) type theory, where we assign types to pure lambda terms. In such a theory, the same term can be assigned multiple inequivalent types. For example, assuming inequivalent types `Bool` and `Nat`, the term $\lambda\, x.\, x$ may be assigned the types $\texttt{Bool} \to \texttt{Bool}$ and $\texttt{Nat} \to \texttt{Nat}$. Church-style (also called *intrinsic*) type theories usually satisfy unicity of typing, by design: a given term has at most one type, modulo type equivalence. In the $\iota\lambda P2$ type theory we consider in this paper, the terms are only the terms of pure lambda calculus; i.e., variables, applications, and lambda abstractions. So we see that unlike the other approaches to inductive types mentioned above, the approach proposed here requires no additional constructs at the term level: terms remain just those of pure lambda calculus. We thus have a solution to the problem of induction in pure type theory (i.e., type theory whose terms are just the pure lambda-calculus terms). Of course, we must make some addition at the type level, or be blocked from deriving induction by Geuvers's result.

The centrality of dependent intersection for induction in $\iota\lambda P2$ is due to its role in internalizing a crucial realizability result of Leivant [28]. He observed that the proofs that data encoded as pure lambda terms using the well-known Church encoding satisfy their typing laws can be identified with those data themselves. In other words, Church-encoded numbers realize their own typings. This remarkable observation is the key