Verifying a Semantic $\beta\eta$-Conversion Test for Martin-Löf Type Theory

Andreas Abel$^1$, Thierry Coquand$^2$, and Peter Dybjer$^2$,*

$^1$ Department of Computer Science, Ludwig-Maximilians-University, Munich
$^2$ Department of Computer Science, Chalmers University of Technology

{coquand,peterd}@cs.chalmers.se

Abstract. Type-checking algorithms for dependent type theories often rely on the interpretation of terms in some semantic domain of values when checking equalities. Here we analyze a version of Coquand’s algorithm for checking the $\beta\eta$-equality of such semantic values in a theory with a predicative universe hierarchy and large elimination rules. Although this algorithm does not rely on normalization by evaluation explicitly, we show that similar ideas can be employed for its verification. In particular, our proof uses the new notions of contextual reification and strong semantic equality.

The algorithm is part of a bi-directional type checking algorithm which checks whether a normal term has a certain semantic type, a technique used in the proof assistants Agda and Epigram. We work with an abstract notion of semantic domain in order to accommodate a variety of possible implementation techniques, such as normal forms, weak head normal forms, closures, and compiled code. Our aim is to get closer than previous work to verifying the type-checking algorithms which are actually used in practice.

1 Introduction

Proof assistants based on dependent type theory have now been around for about 25 years. The most prominent representative, Coq [INR07], has become a mature system. It can now be used for larger scale program development and verification, as Leroy’s ongoing implementation of a verified compiler shows [Ler06]. Functional programmers have also become more and more interested in using dependent types to ensure program and data structure invariants. New functional languages with dependent types such as Agda 2 [Nor07] and Epigram 2 [CAM07] enjoy increasing popularity.

Although many questions about properties of dependent type theories have been settled in the 1990s, some problems are still waiting for a satisfactory solution. One example is the treatment of equality in implementations of proof

* Research partially supported by the EU coordination action TYPES (510996) and the project TLCA of Vetenskapsrådet.
assistants. When we check that a dependently typed program is well-typed, we may need to test whether two types are definitionally equal (convertible). Although it is of course impossible for a system to recognize all semantically equal types, a user will feel more comfortable if it can recognize as many as possible. Whenever it fails the user has to resort to proving manually that the types are equal. This has the additional drawback of introducing proof-objects for these equalities. In recent years there has therefore been a move from \(\beta\)-equality (computational equality) to the stronger \(\beta\eta\)-equality (computational and extensional equality).

Recently, algorithms for testing \(\beta\eta\)-equality have been formulated and verified by the authors both for an untyped notion of conversion \cite{AAD07} and for typed equality judgements \cite{ACD07}. These algorithms use the technique of normalization by evaluation (NbE). However, the algorithms used by proof assistants such as Agda and Epigram \cite{CAM07}, use Coquand’s \(\beta\eta\)-conversion test for semantic ”values”, and do not employ the NbE-technique of the above-mentioned papers. Moreover, there is a gap between algorithms on paper and their actual implementation. Proofs on paper are often informal about the treatment of variable names, and they tend to represent values as pieces of abstract syntax. Besides Pollack’s \cite{Pol94}, Coquand’s algorithm \cite{Coq96} is a notable exception: values are represented as closures, and the algorithm explicitly deals with \(\alpha\)-equivalence by replacing variables by numbers (de Bruijn levels).

We here continue the work of the second author and verify an implementation of the \(\beta\eta\)-conversion test close to the one used in practice. In particular:

- Equality is checked incrementally, and not by full normalization followed by a test for syntactical identity.
- The representation of values is abstract. We only require that they form a syntactical applicative structure. In this way, several possible implementations, such as normal forms, closures, and abstract machine code, are covered by our verification.
- The verification approach is extensible: Although we only spell out the proofs for a core of type theory with predicative universes, our development extends to richer languages. We can for example include a unit type, \(\Sigma\) types, proof irrelevance, and inductive types with large eliminations.

Overview. In Sec. 2 we present an abstract type and equality checking algorithm, which only assumes that the domain of values forms a syntactical applicative structure. In Sec. 3 inference rules for typing and type equality are given for a version of Martin-Löf type theory with explicit substitutions. An outline of the verification is given in Sec. 4 together with a definition of contextual reification, our main tool for verification. Using contextual reification, an alternative equality test can be formulated, which is shown complete in Sec. 5 by construction of a Kripke model, and proven sound in Sec. 6 via a Kripke logical relation. Completeness of the original algorithm then follows easily in Sec. 7. For soundness,