The Inadequacy of Pure Intention

Ryan Wisnesky Paul Govereau

Harvard School of Something

{ryan,govereau}@eecs

Abstract

How to reason about two things when they are only one.

Categories and Subject Descriptors CR-number [*subcategory*]: third-level

General Terms extensionality, type theory

Keywords extensionality, types, theorem proving, identity types, equality

1. Introduction

Every type theory requires an underlying meta-theoretic notion of *definitional* equality, which we denote by $\equiv$. This equality is used by a type-checker to decide the validity of an instance of judgement.

Sometimes the phrase "substitute equals for equals" will implicitly refer to definitional equality; for instance, when definitional equality is reflected into a type theory with a substitution judgement:

$$\textsc{Substitution}\quad \frac{\Gamma \vdash a \qquad \Gamma \vdash b \qquad \Gamma \vdash c \qquad a \equiv b}{\Gamma \vdash c[a \mapsto b]}$$

When type theory is used as a basis for theorem proving it is desirable to have another notion of equality: so called *propositional* equality. Under the Curry-Howard Isomorphism, types are theorems and terms are proofs. If we want to reason about equality, we need types that represent theorems of equality, and terms that represent proofs of these theorems. We denote the type representing a proof that a is equal to b as $a = b$ and refer to it as propositional equality.

These two notions of equality, definitional and propositional, are sometimes referred to as *intentional* equality and *extensional* equality, respectively. We will address the precise meaning and motivation for these terms in a later section. For the moment, suffice it to say that if two terms are definitionally equal, then from a meta-theoretic point of view, they are exactly the same term in all respects. On the other hand, propositionally equal terms are meant to be equivalent with respect to all their properties; they are extensionally equivalent in the sense of Leibniz. Of course, this extensional equivalence crucially depends on the propositional terms behaving within the theory in a way that is consistent with our notions of extensionality.

There is a strong connection (in our minds if nowhere else) between definitional and propositional equality. In the context of theorem proving, definitional equalities can be thought of as theorems that require no proof. That is, we do not have to give terms witnessing their truth because the underlying type theory will automatically apply substitutions and conversions when building up proofs of other facts. In light of the above, we can see that when a type theory is used as the basis for real world applications, stronger formulations of $\equiv$ make it easier for users to work within the theory. This is because the type-checker requires less "help" in the form of manually constructed terms. That is, a powerful type-checker with strong definitional equality can reduce the size of terms and typing derivations given by the user. Unfortunately, strengthening definitional equality too much can make type checking undecidable.

This paper explores the effect that a desire for extensional reasoning has had on the design of definitional equality for a family of type theories used often computer science. The paper proceeds by first defining a prototypical type theory to fix such things as syntax for the purposes of presentation. This theory is then extended to extensional type theory (ETT), which has a very strong internal equality, but is undecidable. We then go on to describe a different extension that is decidable but is inadequate: it fails to preserve canonical forms and therefore fails to be a computational theory in a sense that is useful for computer science. Finally, we describe two different concrete implementations of theories that restore adequacy. In the conclusion, we touch on a different approach, *algebraic* type theory and compare our findings.

2. A Basic Type Theory

We take as our basic type theory the Extended Calculus of Constructions (Luo 1989), presented as a Pure Type System (Barendregt 1992). The complete set of typing rules for our basic type system is given in Figure 1.

The typing rules for ECC are standard for a pure type system. We draw your attention to the (CONVERSION) rule. It is from this rule that definitional equality derives its power. As we see from the rule, if two terms are definitionally equal, then they can be freely substituted for one another in the types of terms. Because of this rule, type checking depends on the decidability of definitional equality.

We will extend this system in various way to explore definitional and propositional equality and their relationship. As we extend the basic system, we would like to preserve the following properties.

Decidability of type checking. We would like type checking of our systems to be decidable. This requires that definitional equality be decidable. In our basic type theory, we take $\equiv$ to be syntactic equality modulo some context-independent, strongly-normalizing reduction strategy. A *context independent* property holds in all (possibly inconsistent) well-typed contexts. Strong normalization of the reductions implies the decidability of type checking. This is because to check the equality of two terms we need only to normal-

$$\text{(ECON)}\ \frac{}{\vdash \{\}} \qquad \text{(ICON)}\ \frac{\Gamma \vdash A : s \qquad s \in \mathcal{S}}{\vdash \Gamma, x : A} \qquad \text{(TYPE)}\ \frac{}{\vdash \mathtt{Type_i} : \mathtt{Type_{i+1}}} \qquad \text{(PROP)}\ \frac{}{\Gamma \vdash \mathtt{Prop} : \mathtt{Type_0}} \qquad \text{(UNIV)}\ \frac{\Gamma \vdash Type_i}{\Gamma \vdash Type_{i+1}} \qquad \text{(VAR)}\ \frac{\vdash \Gamma \qquad \Gamma(x) = A}{\Gamma \vdash x : A}$$

$$\text{(PRODUCT)}\ \frac{\Gamma \vdash A : s \qquad \Gamma, x : A \vdash B : s' \qquad (s, s') \in \mathcal{R}}{\Gamma \vdash \forall x : A.B : s'} \qquad \text{(APP)}\ \frac{\Gamma \vdash t : \forall x : A.B \qquad \Gamma \vdash u : A}{\Gamma \vdash t\ u : B\{u \mapsto x\}} \qquad \text{(LAMBDA)}\ \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x : A.t : \forall x : A.B}$$

$$\text{(CONVERSION)}\ \frac{\Gamma \vdash t : A \qquad \Gamma \vdash B : s \qquad \Gamma \vdash A \equiv B}{\Gamma \vdash t : B}$$

Figure 1. The Extended Calculus of Constructions

ize each term (guaranteed to terminate) and then compare the terms syntactically (decidable). A common choice for the underlying reduction strategy is $\alpha\beta$-reduction.

Adequacy. A type theory is adequate if every term is definitionally equivalent to its canonical form. For example, in the particular case of natural numbers, adequacy would require that every term of type nat is either zero or the successor of a natural number. Put another way, a theory is adequate if every syntactic member of a type has one of that type's constructors as its outermost functional application; this is important because it allows a computer to automatically destruct terms of every type. When adding prepositional equality to our basic system, it is possible to break this property (Hofmann 1995); we discuss this in more detail in Section 5.

2.1 Adding Propositional Equality

So far, our basic theory has no way to reason, within the theory, about proofs of equality. We cannot write down a proposition corresponding to $a = a$, or any other equality in this theory. This is because $\equiv$ is not a predicate in the object-language; it is a meta-theoretic notion. Thus, at this point we will extend our basic theory with introduction rules for propositional equality; all of the systems in this paper will share these two introduction rules:

$$\text{(IDINTRO)}\ \frac{\Gamma \vdash t : A \qquad \Gamma \vdash u : A}{\Gamma \vdash t = u : \mathtt{Prop}} \qquad \text{(REFL)}\ \frac{\Gamma \vdash t : A}{\Gamma \vdash \mathtt{refl}\ t : t = t}$$

The first rule (IDINTRO) introduces the propositional equality type. Note, at this point, a propositional equality can only be formed for two terms of the same type. The second rule (REFL) creates a term witnessing a proof of equality, namely that every term is propositionally equal to itself.

We have not specified any elimination rules for $=$, as we wish to explore several possibilities. However, before doing so we outline several properties that we would like our propositional equality to satisfy.

Propositional Equality is Extensional. Our propositional equality should only equate terms that are extensionally equivalent; that is, they are observationally equal. More formally, we only want to prove propositional equalities of terms that cannot be distinguished by any predicate: all observations on the terms must agree.

Propositional Equality is Substitutive. If we have a proof that two terms contain propositionally equal sub-terms, we would like to be able to coerce one to the other, without changing the computational meaning of the term. This will allow us to use propositional equality in a way similar to definitional equality.

Equality Proofs are Unique. In order to ensure adequacy, we must require that identity proofs are unique (Hofmann and Streicher 1994). In fact, it is enough to say that any two proofs of the same propositional equality are extensionally equivalent—that is, they are themselves propositionally equivalent. Thus, we can restate this requirement by saying that the following type must be inhabited:

$$\forall p\, q : a = b.\ p = q$$

This uniqueness property implies proof irrelevance for propositional equality types. Furthermore, using this property and the substitutive property above, we can freely use propositional equality proofs to rewrite terms without changing their extensional behaviours.

2.1.1 An instance of the basic theory: Coq without axioms

The underlying meta-theory of the theorem prover Coq is an instance of our basic theory. Hence, Coq without any axioms is itself an instance of our basic theory. In Coq, propositional equality is an equivalence relation, named `eq`, defined in the theory with type:

$$\mathtt{eq} : \forall T : Type,\ a\ b : T,\ Prop$$

Propositional equality is not special in any way (relative to any other proposition). The only constructor for `eq` is `refl_equal` which represents proofs by reflexivity. Written in the syntax of Coq, the definition of prepositional equality appears as:

```
Inductive eq (T:Type) (x:T) : T -> Prop :=
    refl_equal : eq T x x.
```

Because `refl_equal` is the only constructor for propositional equality, definitionally equal things can be considered propositionally equal. Note, however, that it is not the case that propositionally equal things are definitionally equal.

In practice, being an equivalence relation is a strong property to ask of a predicate, and the Coq rewriting tactics leverage this strength very well. The extensive set of re-write tactics in Coq dealing with propositional equality often give the illusion that propositional equality *is* definitional equality because one often wants to prove theorems about propositional equality anyway. In general though, Coq tactics need not be sound, only useful, because the type checker verifies correctness of derivations.

2.2 Properties of the basic theory

Our basic theory (and indeed Coq without axioms) satisfies only some of our desired properties. The properties satisfied are shown in Table 1.

The theory is decidable and adequate by construction. The propositional equality is extensional in that it is only inhabited

Decidable	✓
Adequate	✓
Propositional Equality is Extensional	✓
Substitutive	×
Uniqueness of Identity	×

Table 1. Properties of Basic System

by roofs of reflexivity. Therefore, in consistent contexts, propositionally equal terms are also definitionally equal, and thus are observationally equivalent. The last two properties fail to hold in our basic system.

The failure of the Substitutive property goes right the the heart of extensional reasoning. That is, we would like to be able to reason about propositionally equal things using substitution, and this basic system does not give us this power. In the remainder of this paper we will try to restore these last two properties.

3. Mathematical Equality and Extensionality

The word *extensionality* arises is many different contexts and can mean many different but potentially related things. Extensionality is often equated with the lambda-calculus η-rule or functional extensionality because of the central place that functions occupy in computer science and type theory. But the concept of extensionality is actually broader than just the η-rule. For instance, set theory comes with its own definition of extensionality for sets, stating that two sets are equal when they each have the same members. Other mathematical objects may have many different, potentially conflicting notions of extensionality. In fact, broadly speaking, you could define extensional reasoning to be "that which mathematicians actually do." This section informally describes and justifies the fuzzy concept of extensional reasoning.

3.1 What does it mean to do mathematics

From a computer science perspective, mathematicians employ a specific methodology: to describe a class of mathematical objects, they first define a set of syntactic terms to serve as *carriers* for the platonic objects in question, and then form the quotient of that class of terms by some equivalence relation. The particular relation used captures some semantic notion of what it means for the objects denoted by the terms to be equal in the sense that we care about. Having done this, reasoning then proceeds as though the carriers and the equivalence classes of carriers *are actually the same.* In effect, reasoning is done on representatives from each equivalence class.

From a more philosophical perspective, this kind of mathematical reasoning is justified because it employs Leibniz's definition of equality. Leibniz's equality states that one is justified in treating objects identically when there is no test that can distinguish them. By defining a particular equivalence relation when we formed our quotient, we are in effect giving up the ability to reason about properties we think are irrelevant, and we gain a simplifying assumption about our subject.

3.2 An example of extensionality in mathematics

It is natural to give an example of extensional reasoning in mathematics using lambda calculus, but this kind of reasoning is broadly applicable to other areas of mathematics. So, without further ado, let us reason extensionally about lambda terms.

Many mathematical treatments of lambda-calculus are motivated by the desire to reason about operational reduction behavior of lambda terms, and as such, these treatments will often equate α-equivalent terms. On a blackboard in front of a class, this might be done by writing an axiom:

$$\forall t\, t', \alpha(t,t') \Longrightarrow t = t'$$

where α is a predicate capturing the notion of α-equivalence. The development then proceeds without mention of α. This axiom is true because we are only interested in operational behavior, and no operational test can distinguish α-equivalence.

As another example, suppose one is reasoning about simply-typed lambda calculus and lambda terms are considered as set-theoretic functions. In this setting, the pertinent information about a lambda term is captured entirely by the function/lambda term's map, i.e. the set

$$\{(a,b) \mid f\ a = b\} \qquad .$$

To capture this notion, one may form a quotient with the η-equivalence relation. As an axiom, this reads:

$$\forall f\ g\ x, f\ x = g\ x \Longrightarrow f = g \qquad .$$

Of course, there are other possible extensionality principles for functions; the typical example is when the properties in question relate to the running time of functions. If we are interested in these properties, then of course the extensionality axiom above is patently false.

3.2.1 A Philosophical Aside

The use of axioms in this way is said to capture extensional reasoning because the axioms assert that objects are equal based on their "extent", or on *properties each object possesses*, rather than on *the syntactic name or description of the object*. Intensional properties, on the other hand, are ascribed to an object simply by virtue of how the object is named.

The α-equivalence of two terms is in fact decidable based solely on the syntax of a term, whereas the η-equivalence of two terms is not. As such, η-equivalence feels more extensional than alpha-equivalence. In fact, one could imagine creating a type theory with a judgement scheme capturing $alpha$-equivalence of terms. Hence, α-equivalence could be called intensional. However, there is no complete and decidable theory with a judgement scheme for η-equivalence, and so η is truly extensional. We would expect this because the notion of operational reduction is an external concept that we apply to lambda terms.

Broadly speaking, extensional equality can be thought of as denotational equality, and definitional equality as syntactic equality. As one often has a denotation in mind when using a formal system, it is almost always stronger to reason using an extensional notion of equality.

3.3 Extensionality though axioms or judgements

To proceed further we will fix our attention to the function extensionality, as one specific instance of extensionality. This is simply because most work on extensionality is done in the context of functional extensionality.

What we would like our theory to have is an inhabitant of the function extensionality type; our basic theory does not have this. We can get this inhabitant by either having a set of judgements that imply that this type is inhabited (the ETT approach), or we may assume the existence of this inhabitant by using an axiom (the ITTe approach).

4. ETT

Extensional type theory takes a judgemental approach. It adds an equality reflection judgement:

(Equality Reflection)
$$\frac{\Gamma \vdash p : a = b}{\Gamma \vdash a \equiv b}$$

This judgement implies that propositional and definitional equality are the same notion: they may be freely converted to one another by this judgement and by the `refl` constructor.

4.0.1 Undecidability

While this approach is quite powerful, it has the unfortunate drawback of typically making the entire type theory undecidable. A theory with this judgement is called an *extensional type theory.*

One reason that this judgement makes type-checking undecidable is because it requires the type checker to invent the proof p used in the premises of the equality reflection judgement. This is, of course, undecidable in general.

4.0.2 Failure of Strong Normalization

The equality reflection judgement also breaks strong normalization in inconsistent contexts. (It maintains strong normalization for consistent contexts). For example, consider the following example:

```
p: nat->nat = nat
(fun x:nat->nat => x x): (nat->nat)->nat
---------------------------------
(fun x:A => x x)
  (fun x:A => x x) ?: A
```

Here, we can use equality reflection to obtain the equivalent context and goal:

```
p: nat->nat = nat
(fun x:nat->nat => x x): (nat->nat)->nat
(fun x:nat->nat => x x): nat->nat
---------------------------------
(fun x:nat->nat => x x)
  (fun x:nat->nat => x x)
    : nat (by application rule)
```

The extra type for the anonymous function is obtained by equality reflection on p, which has the effect of rendering the application of one function to another well-typed. This term will loop forever when beta-reduced. Therefore to halt, the type-checker would have to test for non-termination, and thus checking is undecidable.

The failure of strong normalization in inconsistent contexts has interesting ramification for the proof theory of extensional type theory. In an extensional type theory, it is inappropriate to view a term as a proof, because the term might not normalize. Instead, one must view the entire typing derivation as the proof, and even then this proof is contingent on the consistency of the axioms involved. Thus, a term isn't a true proof object but is rather a proof object for the partial correctness of the term it types. In spite of non-normalization, however, the theory, interpreted as a logic, is consistent. A term of type False can only occur inside of an inconsistent contexts: an analytic proof of False in a consistent context requires an introduction rule for False. The other two methods for obtaining False are to pull it from the context or to use a non-terminating term; both of these methods require inconsistency to begin with. It is interesting to note that this treatment of proof is in contradiction to the Curry-Howard isomorphism.

In summary, Extensional Type Theory (ETT) has all of the desirable reasoning properties that we desire, but the resulting theory is not decidable. Decidability is such an important property for the applications found in computer science, that we are compelled to abandon ETT to develop an alternative theory.

Decidable	×
Adequate	✓
Propositional Equality is Extensional	✓
Substitutive	✓
Uniqueness of Identity	✓

Table 2. Properties of Extensional Type Theory

5. ITTe

Extensional type theory has all of the reasoning principles that we desire, but type checking is undecidable. This undecidability comes directly from the (Equality Reflection) rule: it is too powerful. The (Equality Reflection) rule requires the type checker to find proofs of propositions in order to decide definitional equality. This "finding of proofs" is not, in general, a solvable problem, and thus the whole theory becomes untenable.

In this section we will develop a decidable type theory with the properties we desire by adding more precise judgments. The resulting system will have all of the properties that we desire, including decidability. To begin, we simply add judgments for functional extensionality and uniqueness of identity:

(Ext)
$$\frac{\Gamma \vdash u, v : \forall x : A.B \qquad \Gamma, x : A \vdash p : u\ x = v\ x}{\Gamma \vdash \mathtt{Ext}(u, v, p) : u = v}$$

(Unique)
$$\frac{\Gamma \vdash u, v : \sigma \qquad \Gamma \vdash p : u = v}{\Gamma \vdash p = \mathtt{refl}\ u : u = v}$$

The first rule, (Ext), give us additional reasoning power. The (Unique) rule is required for adequacy. Our basic system does not have an elimination rule for propositional equality. However, when we add one we must be sure that it cannot be used to produce irreducible terms that are not canonical: i.e. we must ensure the theory remains adequate. The easiest way to do this is to simply require that all proofs of the same propositional equality are in fact definitionally equivalent. This is precisely what the (Unique) judgement does for us.

In addition to ensuring adequacy, the (Unique) rule allows us to give a simpler elimination rule for propositional equality (Hofmann 1993). This simpler elimination rule will give us a way to use propositional equality, and directly ensure the substitutive property.

5.1 Elimination by subst

When looking for an eliminator for propositional equality, we would like to get as much expressive power as possible without sacrificing decidability. In the limit, we would like to have as much expressive power as ETT. Using ETT as a guide, we can derive an elimination rule for propositional equality[1]

In essence, we need to find a decidable equivalent for the use of (Equality Reflection) in our proof trees. That is, we must find a replacement for:

$$\frac{\Gamma \vdash t : U \qquad \dfrac{\Gamma \vdash p : U = T}{\Gamma \vdash U \equiv T}}{\Gamma \vdash t : T}\ .$$

The difficulty with this proof fragment is that the type checker must invent the proof term p during type checking. Therefore, one solution is to simply require that the programmer supply this proof.

[1] This elimination rule is normally derived from the J operator (Altenkirch 1999), this presentation hopes to simplify the issues involved.

To this effect, we introduce a new term `subst` with the following type:

$$\texttt{subst} : \forall a\, b : T.\, (a = b) \to \tau(a) \to \tau(b)$$

The `subst` term carries both a term $\tau(a)$ and a proof $(a = b)$ of propositional equality. Note, that the result type of `subst` gives us the substitution within τ. Using `subst`, we can replace the above proof fragment with the following, decidable fragment:

$$\text{(SUBST)}\quad \frac{\Gamma \vdash t : U \qquad \Gamma \vdash p : U = T}{\Gamma \vdash \texttt{subst}(p, t) : T}$$

Since the proof term p is provided by the programmer as part of the term, the type checker does not need to guess this proof term.

We call the our base system extended with (EXT), (UNIQUE), and (SUBST) extended intentional type theory, or ITTe. This theory has all of the properties that we desire.

Decidable	✓
Adequate	✓
Propositional Equality is Extensional	✓
Substitutive	✓
Uniqueness of Identity	✓

Table 3. Properties of Extended Intentional Type Theory

Note that without unqiueness of identity, we would not have adequacy.

Even though we have the properties we want, we still do not know how expressive the system is. It is possible to give an algorithm which translates terms in ETT into terms in ITTe (Oury 2005); of course, this algorithm is not complete, because non-normalizing terms from ETT have no equivalents in ITTe. In spite of this, however, the same types are inhabited in both ITTe and ETT. This does beg the question: which ETT terms can be encoded in ITTe? We will explore this question in the next section.

5.2 Relation to ETT

Here we state that precise relationship between ITTe and ETT (the following is adapted from Hofmann (1996, 1995)). In order to distinguish judgements of ITTe from ETT, we will write ETT judgments with a subscript ($\vdash_E$).

In constructing ITTe, we added the terms `subst` and `Ext`. In order to compare this system to ETT, we define "strip" (written as $\lfloor t \rfloor$) as a way of removing these extra terms. The complete definition of strip is given below:

$$\lfloor \texttt{Ext}(u, v, p) \rfloor = p$$
$$\lfloor \texttt{subst}(p, t) \rfloor = t$$

The first property that we would like to establish is that ITTe is sound with respect to ETT.

Theorem 5.1 (Soundness)

$$\Gamma \vdash J \Longrightarrow \lfloor \Gamma \rfloor \vdash_E \lfloor J \rfloor$$

In this context, soundness means that any well-typed term of ITTe, if stripped, is a well-typed term of ETT. This soundness theorem is proved by a straight-forward induction on derivations.

The expressiveness of ITTe with respect to ETT is established by the following two theorems.

Theorem 5.2 (Conservative Types)

$$\lfloor \Gamma \rfloor \vdash_E \lfloor \sigma \rfloor \Longrightarrow \Gamma \vdash \sigma$$

Theorem 5.3 (Conservative Propositional Equality)

$$\lfloor \Gamma \rfloor \vdash_E \lfloor M \rfloor = \lfloor N \rfloor \Longrightarrow \Gamma \vdash \sigma M = N$$

The first theorem states that for any well-formed type in ETT, there exists a type, potentially with some `Ext` and `subst` terms inserted, that is well formed in ITTe. The second theorem is much more interesting. The second theorem establishes that and proof of propositional equality in ETT has a corresponding proof in ITTe when the types make sense. This last result gives us a good deal of confidence that our system is capable of expressing all of the propositional equalities that we many want.

5.2.1 A Surjective Embedding

Notice, that not every well-typed term in ETT has a corresponding stripped version in ITTe. That is, the embedding of ETT into ITTe is not surjective. For example, the following judgement in ETT:

$$x : nat \vdash_E x = S^x(0)$$

One possible embedding of this into ITTe is:

$$x : nat \vdash subst(.., x) = S^x(0)$$

However, this is not provable in ITTe: currently we have no way to reason about the propositional equality of `subst` terms. Hofmann (1995) proposed a solution which involves adding an reasoning principle. The new reasoning principle would provide a term in ITTe which is definitionally equal to $S^x(0)$ in ITTe, but when stripped would be definitionally equal to x in ETT.

Adding these new reasoning principles, at all of the necessary types, would allow us to construct, for every judgment in ETT, a corresponding judgment in ITTe. We would then have a fully surjective embedding of ETT into ITTe. We do not go further into the details here, we only mention this because is bears some similarity to the techniques used in Observational Type Theory (Altenkirch and McBride 2006).

6. Alternative Approaches

In this presentation, we have axiomatized uniqueness. There are a number of axioms which are equivalent to uniqueness of identity. To mention a few:

- Invariance by Substitution of Reflexive Equality Proofs.
- Injectivity of Dependent Equality
- Uniqueness of Identity Proofs
- Uniqueness of Reflexive Identity Proofs
- Streicher's Axiom K

Any of the above axioms would suffice. Indeed, any one of these axioms is sufficient to prove all of the others.

Rather that introducing any of the axioms above, we could have taken another route, and formalized heterogeneous equality (McBride 2000), also know as "John Major" equality. The heterogeneous equality approach is taken in Observational Type Theory (Altenkirch and McBride 2006). In addition, OTT introduces a constructor for equality at each type in the system. This gives them a more expressive type theory, but makes establishing proof irrelevance challenging. At the time of this writing, OTT looks like a very promising alternative, but the meta-theory has not been fully worked out.

In this paper we have focused on type theoretic solutions. However, there are other approaches that can be taken. From a practical point of view, the problem with ITT is that we often encounter terms that are equivalent, but not definitionally equal. For example, in Coq $n + 0$ is definitionally equal to n, but $0 + n$ is not. The programmer (prover) can perform a sequence of rewritings to transform $0 + n$ into $n + 0$, and then definitional equality takes over. Hence, we could consider adding a set of confluent term rewriting equations to the system which automates these kinds of tasks. This generally goes by the name of "Algebraic Type Theory". The primary difficulty with Algebraic Type Theories is that it is easy to accidentally destroy the meaning of your types by introducing rewritings.

7. Conclusion

In this paper, we have begun the process of outlining the difficulties of propositional equality and some of the current solutions. In our presentation we have focused on five properties that motivate the solutions discussed. These are certainly not the only properties of interest, or the solutions possible. However, we hoped to bring a narrative cohesion to an otherwise complex and varying field of study.

References

Thorsten Altenkirch. Extensional equality in intensional type theory. In *LICS '99: Proceedings of the 14th Annual IEEE Symposium on Logic in Computer Science*, page 412, Washington, DC, USA, 1999. IEEE Computer Society. ISBN 0-7695-0158-3.

Thorsten Altenkirch and Conor McBride. Towards observational type theory. University of Nottingham, 2006.

Henk Barendregt. Lambda calculi with types. In *Handbook of Logic in Computer Science, Volumes 1 (Background: Mathematical Structures) and 2 (Background: Computational Structures), Abramsky & Gabbay & Maibaum (Eds.), Clarendon*, volume 2. 1992.

Martin Hofmann. Elimination of extensionality in martin-löf type theory. In Henk Barendregt and Tobias Nipkow, editors, *TYPES*, volume 806 of *Lecture Notes in Computer Science*, pages 166–190. Springer, 1993. ISBN 3-540-58085-9.

Martin Hofmann. Conservativity of equality reflection over intensional type theory. In *TYPES '95: Selected papers from the International Workshop on Types for Proofs and Programs*, pages 153–164, London, UK, 1996. Springer-Verlag. ISBN 3-540-61780-9.

Martin Hofmann. *Extensional concepts in intensional type theory*. PhD thesis, University of Edinburgh, 1995.

Martin Hofmann and Thomas Streicher. The groupoid model refutes uniqueness of identity proofs. In Samson Abramsky, editor, *Proceedings of the Ninth Annual IEEE Symp. on Logic in Computer Science, LICS 1994*, pages 208–212. IEEE Computer Society Press, July 1994.

Zhaohui Luo. ECC, an extended calculus of constructions. In *Proceedings 4th Annual IEEE Symp. on Logic in Computer Science, LICS'89, Pacific Grove, CA, USA, 5–8 June 1989*, pages 386–395. IEEE Computer Society Press, Los Alamitos, CA, 1989.

Zhaohui Luo. *An Extended Calculus of Constructions*. PhD thesis, University of Edinburgh, 1990. URL `citeseer.ist.psu.edu/luo90extended.html`.

Conor McBride. *Dependently Typed Functional Programs and Their Proofs*. PhD thesis, 2000.

Nicolas Oury. Extensionality in the calculus of constructions. In *TPHOLs*, pages 278–293, 2005.

Frank Pfenning. Intensionality, extensionality, and proof irrelevance in modal type theory. In *LICS '01: Proceedings of the 16th Annual IEEE Symposium on Logic in Computer Science*, page 221, Washington, DC, USA, 2001. IEEE Computer Society.