Master thesis proposal: Erasing propositions in homotopy type theory

Gabe Dijkstra

April 4, 2013

1 Introduction

In this thesis we concern ourselves with the following notion of equality, the so called *identity types* as introduced by Martin-Löf (1985). In Agda they can be defined as follows:

definition}

```
data Id (A : Set) (x : A) : A → Set a where
  refl : Id A x x
```

In Martin-Löf's system, the only way to do something with data, is via elimination operators (also called induction principles): there is no pattern matching. If we allow for pattern matching, we can easily proof that if we have two proofs $p, q : Id\ A\ x\ y$, then these proofs are also equal, i.e. we have a proof $r : Id\ (Id\ A\ x\ y)\ p\ q$. This property is also called *uniqueness of identity proofs* (UIP).

However, we cannot prove UIP from the elimination operator of the identity types (also called J), as has been shown by Hofmann and Streicher (1996). This means that pattern matching is a non-conservative extension over Martin-Löf's system.

In fact, the structure that arises from the identity types looks a lot like the homotopy groupoids from homotopy theory. If we interpret a type A as a topological space (up to homotopy) and $x, y : A$ as points in this space, then a proof $p : Id\ A\ x\ y$ corresponds to a path from x to y. This interpretation of types leads to interesting models of intensional type theory and is the subject of homotopy type theory.

2 Contributions

2.1 Introduction to homotopy type theory

There are several introductions to homotopy type theory (e.g. Awodey (2012) and Pelayo and Warren (2012)), but these are geared towards mathematicians who know about homotopy theory, but do not know about type theory. For the computer scientist who knows some type theory, but has never seen any homotopy theory, there is virtually no material.

> **Contribution:** We provide an introduction to homotopy type theory for the computer scientist who has some familiarity with type theory, but does not have the background in homotopy theory.

2.2 Erasibility of propositions

One way to explain type theory is using the propositions-as-types analogy, also called the Curry-Howard correspondence. In practice, this identification is not precise enough. If we have two proofs of a proposition, we tend to regard those proofs as equal. We care more about the fact that we have a proof rather than how the proof is constructed exactly. However, there are types where we definitely do not want to regard all inhabitants to be equal, for example the natural numbers.

Instead of viewing all types as propositions, we identify propositions only with those types whose inhabitants are all (propositionally) equal to eachother:

$$\begin{aligned} &\mathit{isProposition} : \mathit{Set} \to \mathit{Set} \\ &\mathit{isProposition}\ A = (x\ y : A) \to \mathit{Id}\ A\ x\ y \end{aligned}$$

In homotopy type theory a type that has this property is usually called *(-1)-truncated* or a *(-1)-type*.[1]

Examples of propositions are the empty type $\bot$ and the unit type $\top$. In fact, we can prove that if a type is inhabitated and it is a proposition, it is isomorphic to the unit type.

2.2.1 Comparison with collapsibility

The definition $\mathit{isProposition}$ looks a lot like *collapsibility* (Brady et al., 2004). Given some indexing type I, a family $D : I \to \mathit{Set}$ is called *collapsible* if for all indices i and inhabitants $x, y : D\ i$, x and y are definitionally equal. In other words: every $D\ i$ is either empty or has one element (up to definitional equality).

[1] The somewhat strange numbering comes from homotopy theory, where 0-truncated intuitively means that we have a set, (-1)-truncated a set of at most one element and (-2)-truncated an empty set

If we know that a family is collapsible, we can optimise its constructors and elimination operators by erasing certain parts, since we know that the relevant parts (if there are any) can be recovered from the indices.

Comparing the definition of collapsible families to $isProposition$, we notice that they are largely the same. $isProposition$ is a can be seen as an internalised version of collapsibility: we have replaced definitional equality with propositional equality.

A question one might ask is whether the optimisations based on collapsibility also hold for propositions. If our type theory satisfies canonicity, propositional equality implies definitional equality in the empty context. This means, that we can indeed use the concept of propositions for the same optimisations as those for collapsible families.

In homotopy type theory, one usually adds axioms such as the univalence axiom, or axioms to implement higher inductive types. This means we lose canonicity hence we no longer have that propositional equality always implies definitional equality.

Not all is lost: let B be a type for which all proofs $p : Id\ B\ x\ y$ are definitionally equal to $refl$ and let A be a (-1)-type, then the only functions we $f : A \to B$ we can write are (definitionally) constant functions, hence we can at run-time ignore what value of A we get exactly.

> **Contribution:** We identify cases where (-1)-types can be safely erased: we provide an optimisation in the spirit of Brady et al. (2004)

2.2.2 Comparison with Prop in Coq

In Coq we can make the distinction between informative or computational parts of our program (everything that lives in Set) and logical parts (everything that lives in $Prop$). This distinction is also used when extracting a Coq development to another language: we can safely erase terms of sort $Prop$.

Another feature of $Prop$ is its impredicativity.

> **Contribution:** We provide a comparison between Coq's $Prop$ universe to the (-1)-types of homotopy type theory.

2.3 Applications of homotopy type theory to programming

2.3.1 Quotients

Higher inductive types provide for a natural construction of quotients. In pseudo-Agda this would look as follows:

$$\begin{aligned}&\textbf{data}\ Quotient\ (A : Set)\ (R : A \to A \to Proposition) : Set\ \textbf{where}\\ &\quad project : A \to Quotient\ A\ R\\ &\quad relate\ \ : (x\ y : A) \to R\ x\ y \to Id\ (Quotient\ A\ R)\ (project\ x)\ (project\ y)\end{aligned}$$

$$\begin{aligned} contr \;\; &: (x\ y : Quotient\ A\ R) \to (p\ q : Id\ (Quotient\ A\ R)\ x\ y) \\ &\to Id\ (Id\ (Quotient\ A\ R)\ x\ y)\ p\ q \end{aligned}$$

Quotienting out by the given relation means that we need to regard two terms x and y related to eachother with R as propositionally equal, which is witnessed by the $relate$ constructor. In order for the result to be a set (in the sense of being a discrete groupoid), we also need to ensure that it satisfies UIP, which in turn is witnessed by the $contr$ constructor.

> **Contribution:** We compare this approach to quotients to other approaches, such as definable quotients (Altenkirch et al.).

2.3.2 Views on abstract types

The univalence axiom should make it more easy to work with views in a dependently typed setting.[2]

> **Contribution:** Make the sketches and computations from Dan Licata's blog post more precise

There are cases where it makes sense to have an implementation that has more structure than we want to expose to the user using the view. Instead of having an isomorphism, we then have a section/retraction pair. Since we have quotients to our disposal, we can make this into an isomorphism.

> **Contribution:** Identify examples where this is useful.

References

Thorsten Altenkirch, Thomas Anberrée, and Nuo Li. Definable quotients in type theory.

Steve Awodey. Type theory and homotopy. In *Epistemology versus Ontology*, pages 183–201. Springer, 2012.

Edwin Brady, Conor McBride, and James McKinna. Inductive families need not store their indices. In *Types for Proofs and Programs*, pages 115–129. Springer, 2004.

Martin Hofmann and Thomas Streicher. The groupoid interpretation of type theory. In *In Venice Festschrift*, 1996.

Per Martin-Löf. Constructive mathematics and computer programming. In *Proc. of a discussion meeting of the Royal Society of London on Mathematical logic and programming languages*, pages 167–184. Prentice-Hall, Inc., 1985.

Álvaro Pelayo and Michael A Warren. Homotopy type theory and voevodsky's univalent foundations. *arXiv preprint arXiv:1210.5658*, 2012.

[2][[http://homotopytypetheory.org/2012/11/12/abstract-types-with-isomorphic-types/][http://homotopytypetheory.org/2012/11/12/abstract-types-with-isomorphic-types/]]