Computational Higher Type Theory (CHTT)

Robert Harper

Lecture Notes of Week 7 by Evan Cavallo

1 Setting the Scene

Last week, we introduced intensional (dependent) type theory (ITT), a formal type theory which adds type-indexed families of types (or *dependent types*) and corresponding type formers. This type theory is defined by way of four inductively defined judgments.

$\Gamma \vdash A\ \mathsf{type}$	A is a type in context Γ.
$\Gamma \vdash A \equiv A'\ \mathsf{type}$	A and A' are equivalent types in context Γ.
$\Gamma \vdash M : A$	M has type A in context Γ.
$\Gamma \vdash M \equiv M' : A$	M and M' are equal elements of A in context Γ.

These judgments satisfy structural properties (weakening, contraction, and exchange), which can either be proven by analyzing the inductive definition or added by axiomatic fiat. Definitional equivalence is a very fine form of "equality", generated by reflexivity, symmetry, transitivity, compatibility with the operations of the type theory, and simplification equations arising from Gentzen's inversion principle. We defined three type formers for ITT: Σ (dependent sum/product) types, Π (dependent product/function) types, and identity types.[1]

This week, we continue to examine the properties of the identity types $\mathsf{Id}_A(M, N)$, which internalize a notion of equality between elements M and N of a type A. In particular, we want to bring out certain unsatisfactory properties of such a notion. We will then move to consider a *computational* dependent type theory defined by behavioral judgments $\Gamma \gg A \doteq A'\ \mathsf{type}$ and $\Gamma \gg M \doteq M' \in A$, in which we have (and can internalize) a behavioral notion of equality of and within types.

2 Identity types and definitional equivalence

Let us recall quickly recall (some of) the rules defining identity types.

$$\text{(F)}\quad \frac{\Gamma \vdash A\ \mathsf{type} \qquad \Gamma \vdash M : A \qquad \Gamma \vdash N : A}{\Gamma \vdash \mathsf{Id}_A(M, N)\ \mathsf{type}} \qquad \text{(I)}\quad \frac{\Gamma \vdash M : A}{\Gamma \vdash \mathsf{refl}_{[}(A)]M : \mathsf{Id}_A(M, M)}$$

$$\text{(E)}\quad \frac{\begin{array}{c}\Gamma, a : A, b : A, p : \mathsf{Id}_A(a, b) \vdash C\ \mathsf{type} \\ \Gamma, a : A \vdash Q : [a, a, \mathsf{refl}_{[}(A)]a/a, b, p]C \qquad \Gamma \vdash P : \mathsf{Id}_A(M, N)\end{array}}{\Gamma \vdash \mathsf{J}_{a,b,p.C}(a.Q)(P) : [M, N, P/a, b, p]C}$$

$$(\beta)\quad \frac{\begin{array}{c}\Gamma, a : A, b : A, c : \mathsf{Id}_A(a, b) \vdash C\ \mathsf{type} \\ \Gamma, a : A \vdash Q : [a, a, \mathsf{refl}_{[}(A)]a/a, b, c]C \qquad \Gamma \vdash M : A\end{array}}{\Gamma \vdash \mathsf{J}_{a,b,c.C}(a.Q)(\mathsf{refl}_{[}(A)]M) \equiv [M/a]Q : [M, N, P/a, b, c]C}$$

[1] TODO "dependent product"

The introduction form $\mathsf{refl}_{[}(A)]M$ can be thought of as expressing that $\mathsf{Id}_A(-,-)$ is a reflexive "relation:" for any $M : A$, we have an element of $\mathsf{Id}_A(M, M)$. The eliminator then expresses that $\mathsf{Id}_A(-,-)$ is the *least* reflexive relation: to define a map out of the family $\mathsf{Id}_A(-,-)$ into a family $a.b.p.C$ (the *motive*), it suffices to consider the reflexive cases. As we suggested last week, this fact can be used to establish that $\mathsf{Id}_A(-,-)$ is also symmetric and transitive, making it an equivalence relation on elements of A. We'll start by proving these facts, so we can get a feel for working with the identity type. For readability, we will now write the type $\mathsf{Id}_A(M, N)$ as $M =_A N$.

Lemma 1. *For all $\Gamma \vdash P : M =_A N$, there exists a term $\Gamma \vdash \mathsf{sym}_A(M; N; P) : N =_A M$.*

Proof. We are defining a map out of the identity type $M =_A N$ into $N =_A M$. As such, we will want to use the identity eliminator. The eliminator requires us to think in terms of the *family* of all identity types in A: given *any* $a, b : A$ and a term $p : a =_A b$, what type $C(a, b, p)$ to we want to inhabit? In this case, it seems sensible to take $C(a, b, p) := b =_A a$. The elimination principle says that to inhabit $C(a, b, p)$ in general, it suffices to inhabit the reflexive case $C(a, a, \mathsf{refl}_{[}(A)]a)$ given $a : A$. We have $C(a, a, \mathsf{refl}_{[}(A)]a) = a =_A a$, so we can inhabit this type with $\mathsf{refl}_{[}(A)]a$. Returning to our original objective, the eliminator now gives us an element of $C(M, N, P)$, that is, $N =_A M$. Written out in full, this element is

$$\mathsf{sym}_A(M; N; P) := \mathsf{J}_{a.b.p.b=_A a}(a.\mathsf{refl}_{[}(A)]a)(P). \qquad \square$$

Let us reflect for a moment on the term we have defined to implement symmetry. By the β rule for the eliminator, we have $\mathsf{sym}_A(M; M; \mathsf{refl}_{[}(A)]M) \equiv \mathsf{refl}_{[}(A)]M : M =_A M$; if we put a reflexivity in, we get a reflexivity out. This is a definitional equivalence, which we can internalize to inhabit the corresponding identity type:

$$\mathsf{refl}_{[}(A =_) \; MM]\mathsf{refl}_{[}(A)]M : \mathsf{sym}_A(M; M; \mathsf{refl}_{[}(A)]M) =_{M=_A M} \mathsf{refl}_{[}(A)]M.$$

Lemma 2. *For every $\Gamma \vdash P : M =_A N$ and $\Gamma \vdash Q : N =_A O$, there exists a term $\Gamma \vdash \mathsf{trans}_A(M; N; O; P; Q) : M =_A O$.*

Proof. Once again, we are defining a map out of the identity type, so we will have to use the eliminator. In this case, we have a choice: we can start by applying the eliminator either with P or with Q. Let's try the former. As before, we have to generalize to the family of identity types: given $a, b : A$ and $p : a =_A b$, what is the type $C(a, b, p)$ we want to inhabit? Here, the answer is a bit less obvious. One option is to take

$$C(a, b, p) := (c : C) \to b =_A c \to a =_A c.$$

We think of $C(a, b, p)$ as the type of functions which extend any $q : b =_A c$ by our identification $p : a =_A b$, returning an identification $a =_A c$.

To use the eliminator, we have to supply the reflexivity case $C(a, a, \mathsf{refl}_{[}(A)]a)$. Expanding the definition of C, we need a function $(c : C) \to a =_A c \to a =_A c$. For this, we can take the identity function $\lambda c{:}C.\, \lambda q{:}a =_A c.\, q$.

Applying the eliminator we have described to the identification P gives us the term

$$\mathsf{J}_{a.b.p.(c:C)\to b=_A c\to a=_A c}(a.\lambda c{:}C.\, \lambda q{:}a =_A c.\, q)(P) : (c : C) \to N =_A c \to M =_A c.$$

We can take $\mathsf{trans}_A(M; N; O; P; Q)$ to be the result of applying this function at O and Q. $\square$

Once again, let's take a look at the definitional equivalences satisfied by the transitivity term. By the β rule for the eliminator, we have

$$\mathsf{trans}_A(M; M; N; \mathsf{refl}_{[}(A)]M; Q) \equiv (\lambda c{:}C.\, \lambda q{:}a =_A c.\, q)(N)(Q) \equiv Q.$$

On the other hand, we *don't* have an equivalence $\mathsf{trans}_A(M; N; N; P; \mathsf{refl}_{[}(A)]N) \equiv P$ in general! This is because we defined the transitivity term by eliminating with P; if we had used Q, the situation would be reversed. Here we see another example of the finicky nature of definitional equivalence. On the other hand, one can (exercise!) inhabit the identity type

$$\mathsf{trans}_A(M; N; N; P; \mathsf{refl}_{[}(A)]N) =_{M=_A N} P.$$

Thus, the identity type provides a notion of equality strictly weaker than definitional equivalence. We might wonder whether it can rightly be called a notion of equality at all: at the moment, we only know it is an equivalence relation. One encouraging property is Leibniz's principle of *indiscernibility of identicals*: any terms related by an identification have the same properties.

Theorem 3 (Indiscernibility of identicals)**.** *Let* $\Gamma \vdash P : M =_A N$. *For any* $\Gamma, a : A \vdash B\ \mathsf{type}$, *there is a function* $\Gamma \vdash \mathsf{transport}[a.B] : [M/a]B \to [N/a]B$.

Proof. Define $\mathsf{transport}[a.B] := \mathsf{J}_{b.c.p.[b/a]B \to [c/a]B}(a.\lambda d{:}B.\, d)(P)$. □

When we read the family B as a predicate (i.e., family of propositions), the type of $\mathsf{transport}[a.B]$ says that the property $[M/a]B$ implies $[N/a]B$. If we apply the theorem again with $\mathsf{sym}_M(N; P;)$, we find that $[N/a]B$ likewise implies $[M/a]B$. So $[M/a]B$ and $[N/a]B$ are logically equivalent: M satisfies B if and only if N satisfies B. As this is true for all B, we can say that M and N satisfy the same predicates.

Sadly, our high hopes for the identity type are dashed when we examine identity at function type. As an example, consider the function $\lambda b{:}\mathsf{bool}.\, \mathrm{if}_{\_.\mathsf{bool}}(b; \mathsf{true}; \mathsf{false})$. As we have already discovered, this function is not definitionally equivalent to the identity function. However, we can construct an identification

$$b : \mathsf{bool} \vdash \mathrm{if}_{b.\mathrm{if}_{\_.\mathsf{bool}}(b;\mathsf{true};\mathsf{false}) =_{\mathsf{bool}} b}(b; \mathsf{refl}_{[}(\mathsf{bool})]\mathsf{true}; \mathsf{refl}_{[}(\mathsf{bool})]\mathsf{false}) : \mathrm{if}_{\_.\mathsf{bool}}(b; \mathsf{true}; \mathsf{false}) =_{\mathsf{bool}} b$$

by cases on the form of b. So far, this looks great: we can use the identity type to establish "equalities" which require non-trivial reasoning, something we couldn't do with definitional equivalence. But suppose we now want to inhabit the type

$$\cdot \vdash (\lambda b{:}\mathsf{bool}.\, \mathrm{if}_{\_.\mathsf{bool}}(b; \mathsf{true}; \mathsf{false})) =_{\mathsf{bool}\to\mathsf{bool}} (\lambda b{:}\mathsf{bool}.\, b)\ \mathsf{type}.$$

It turns out that this is impossible! This fact follows from a theorem of Martin-Löf [1975, Theorem 3.14], which characterizes the inhabitants of closed identity types.

Theorem 4 (Martin-Löf)**.** $\cdot \vdash \mathsf{Id}_A(M, N)\ \mathsf{type}$ *is inhabited if and only if* $\cdot \vdash M \equiv N : A$.

This theorem is a corollary of a normalization result: if P is a closed term inhabiting $\mathsf{Id}_A(M, N)$, then P reduces to a canonical value in $\mathsf{Id}_A(M, N)$. As the only canonical values in $\mathsf{Id}_A(M, N)$ are the reflexivity terms, $\mathsf{Id}_A(M, N)$ can only be inhabited if it is inhabited by some $\mathsf{refl}_{[}(A)]O$, which in turn is only possible when $\cdot \vdash M \equiv N : A$.

Thus, unlike definitional equivalence, the identity type is not compatible with λ-abstraction. We say that intensional type theory lacks *function extensionality*, meaning more specifically that the rule

$$\frac{\Gamma \vdash (a : A) \to \mathsf{Id}_B(Fa, Ga)\ \mathsf{true}}{\Gamma \vdash \mathsf{Id}_{(a:A)\to B}(F, G)\ \mathsf{true}}\ (\textsc{Funext})$$

is not admissible. We call this *extensionality* because it expresses that a function is characterized by its *extension*, that is, its behavior on arguments. In intensional type theory, functions with different *intension*, i.e., differing syntactic definitions, may not be identifiable even if their extensions are equal.

3 Pursuing function extensionality

The lack of function extensionality is quite a blow for intensional type theory, as it prevents us from working with functions in any way resembling traditional mathematics. As such, there are a number of approaches for recovering a form of extensionality.

Go Straight to Setoid Hell If we want to endow $(a : A) \to B$ with the "correct" notion of equality, one option is to toss the identity type aside and instead manually define the equality relations we want. Instead of working with types, we can instead work (still inside ITT) using types $\Gamma \vdash A\ \mathsf{type}$ paired with equivalence relations $\Gamma, a : A, a' : A \vdash R\ \mathsf{type}$. These pairs are called *setoids*. Given setoids $(A, a.a'.R)$ and $(B, b.b'.S)$, for example, we would define their function setoid as having carrier $A \to B$ and equivalence relation

$$f : A \to B, f' : A \to B \vdash (a : A) \to (a' : A) \to R(a, a') \to S(f(a), f'(a'))\ \mathsf{type}.$$

With this relation serving as equality for $A \to B$, we recover extensionality. Unfortunately, passing equivalence relations around for every type quickly becomes prohibitively bureaucratic, as one must check explicitly that every function one defines preserves equivalence.

Extensional Type Theory We know that definitional equivalence is compatible with λ-abstraction, so we might view the problem as the distance between definitional equivalence and identity types. We can try closing this gap by adding an *equality reflection* rule.

$$\frac{\Gamma \vdash P : \mathsf{Id}_A(M, N)}{\Gamma \vdash M \equiv N : A}$$

This rule allows us to transform any inhabited identity type into a definitional equivalence. It is usually accompanied by a rule expressing *uniqueness of identity proofs*, which states that there is at most one inhabitant of a given identity type.

$$\frac{\Gamma \vdash P : \mathsf{Id}_A(M, N) \qquad \Gamma \vdash Q : \mathsf{Id}_A(M, N)}{\Gamma \vdash P \equiv Q : \mathsf{Id}_A(M, N)}$$

The resulting system is often called *extensional type theory* (ETT). Note that these rules are already admissible in ITT when $\Gamma = \cdot$ by theorem 4, so their addition does not disrupt the computational interpretation of closed terms. By unifying definitional equivalence with identity types, we get the desirable properties of both: compatibility with λ-abstraction and the ability to establish equivalence which require non-trivial reasoning.[2] However, the latter can also be seen as a weakness: it means that the definitional equivalence judgment $\Gamma \vdash M \equiv N : A$, and thereby every judgment of the type theory, ceases to be decidable. Terms thus become distinct from derivations: a well-typed term no longer contains all the information necessary to recover its typing derivation, as proofs of identity are forgotten in equality reflection.

New Identity Axioms Finally, we can simply assert function extensionality as a new rule.

$$\frac{\Gamma \vdash P : (a : A) \to \mathsf{Id}_B(Fa, Ga)}{\Gamma \vdash \mathsf{funext}_{A,B}(F; G; P) : \mathsf{Id}_{(a:A)\to B}(F, G)}$$

Unlike in ETT, we are adding identifications rather than definitional equivalences, so there is no effect on the decidability of typing judgments. In this case, however, we disrupt the

[2] To get function extensionality, we also need an η rule stating that $\Gamma \vdash F \equiv \lambda a{:}A.\, Fa : (a : A) \to B$ for any $\Gamma \vdash F : (a : A) \to B$.

computation system: we have no way of reducing a term like

$$\mathsf{J}_{a.b.p.C}(a.Q)(\mathsf{funext}_{A,B}(F;G;P)).$$

In essence, we have added a new canonical form to the identity type without providing a corresponding reduction rule for the eliminator. There do exist solutions to this problem: see the *observational type theory* of Altenkirch et al. [2007].

Homotopy type theory [Univalent Foundations Program, 2013] obtains function extensionality by adding the stronger *univalence axiom*, which provides a form of extensionality for elements of a universe (a type of types). Roughly speaking, univalence states that identities between types in the universe correspond to isomorphisms between those types. Surprisingly, the univalence axiom implies function extensionality for function types formed from types in the universe. However, homotopy type theory suffers from a lack of computational interpretation for the same reason: there is no way to evaluate the identity eliminator on an identification created by univalence. Only recently have variants of HoTT with computational interpretations arrived [Cohen et al., 2016, Angiuli et al., 2017]; we will discuss these in future weeks.

4 Computational dependent type theory

Now that we have the formal system of ITT in hand, we would like to bring out its computational properties. Informally, we want to show that we can read computational objects from well-typed terms, and that these terms can be evaluated to obtain some result. There are thus two components to the computational interpretation, much as in the non-dependent case:

Erasure. The terms of ITT are full of annotations which have no relevance to computation, such as the A in $\mathsf{refl}_{[}(A)]M$ or $\lambda a{:}A.\,N$. (We have already suppressed many of these annotations—$\lambda a{:}A.\,N$ should really be annotated with the target family $a.B$, for example.) Moreover, certain operations like identity elimination have no computational content at all. We therefore want an *erasure* or *extraction* operation $|-|$, which extracts the part of a term which is actually relevant to computation.

Operational semantics. Next, we need to describe the process by which we execute terms. We can encode this as a deterministic *structural operational semantics* on closed erased terms, specified by judgments $M \mapsto M'$ and $M\ \mathsf{value}$.

We also want to relate this interpretation of terms to the types of ITT, interpreting the type of a term as a specification of its erasure's execution behavior. This semantics is called a *meaning explanation* of the types, following Martin-Löf [1982], and corresponds to the hereditary termination and equality predicates we have previously defined. The meaning explanation takes the form of semantic equivalents of each formal judgment:

$$\Gamma \gg A\ \mathsf{type}, \quad \Gamma \gg A \doteq A'\ \mathsf{type}, \quad \Gamma \gg M \in A, \quad \Gamma \gg M \doteq M' \in A.$$

In the non-dependent case, the equivalent of the judgment $\Gamma \gg M \in A$ was defined as $\forall\gamma.\mathsf{HT}_\Gamma(\gamma) \supset \mathsf{HT}_A(\hat{\gamma}|M|)$. In the dependent case, terms can appear in types, so types themselves must be endowed with an extraction operation and an operational semantics. Erased types, then, are simply particular terms, namely those which evaluate to a code for a specification. These include programs like bool and $\mathsf{bool} \to \mathsf{bool}$, which are already values, but also programs like $\mathsf{if}(\mathsf{true};\mathsf{bool};\mathsf{nat})$, which evaluates to the specification bool. Note that the "small" and "large" eliminators of the formal theory can be erased to the same operator.

We are thus led to the following rough reading of the closed semantic judgments.

A type	A is a well-behaved program evaluating to a canonical type.
$A \doteq A'$ type	A and A' evaluate to equivalent canonical types.
$M \in A$	A evaluates to a canonical type, and M evaluates to a canonical value of that canonical type.
$M \doteq M' \in A$	A evaluates to a canonical type, and M and M' to equivalent canonical values of that canonical type.

The basis of a meaning explanation for a type theory is thus the definition of its canonical types and the canonical values of said types. We will regard A type and $M \in A$ as being derived from the other two judgments, with A type meaning $A \doteq A'$ type and $M \in A$ meaning $M \doteq M \in A$. The open judgment forms, $\Gamma \gg A \doteq A'$ type and $\Gamma \gg M \doteq M' \in A$, are defined from the closed judgments by *functionality*: equal open terms are those which take equal closing substitutions to equal closed terms. As an example, we will define the open judgments for the case of a one-element context.

Definition 5.

- *The judgment* $a : A \gg B \doteq B'$ type *holds when for all* $M \doteq M' \in A$*, we have* $[M/a]B \doteq [M'/a]B'$ type.
- *The judgment* $a : A \gg N \doteq N' \in B$ *holds when for all* $M \doteq M' \in A$*, we have* $[M/a]N \doteq [M'/a]N' \in [M/a]B$.

The general judgments $\Gamma \gg A \doteq A'$ type and $\Gamma \gg M \doteq M' \in A$ are defined in this fashion by induction on the length of the context Γ. (For a complete definition of this iterated functionality, see for example Martin-Löf [1982].)

The last step to giving a full meaning explanation for type theory is to explain the equal canonical types and the equal canonical values in each type. As an example, we add booleans to the type theory as follows.

1. bool and bool are equal canonical types.
2. true and true are canonical values satisfying the specification bool.
3. false and false are canonical values satisfying the specification bool.
4. No other values satisfy bool. In other words, bool is inductively generated by the previous two clauses.

From these, it follows trivially that bool type, true $\in$ bool, and false $\in$ bool. But these are not the only facts which follow from this definition: we also get the elimination rule.

Theorem 6. *If*

1. b : bool $\gg A$ type*,*
2. $M \doteq M' \in$ bool*,*
3. $N \doteq N' \in [\mathsf{true}/b]A$*, and*
4. $P \doteq P' \in [\mathsf{false}/b]A$*,*

then $\mathrm{if}(M; N; P) \doteq \mathrm{if}(M'; N'; P') \in [M/b]A$.

This theorem follows by an analysis of the operational behavior of the conditional terms, using what $M \doteq M' \in$ bool tells us about the behavior of M and M'.

Next week, we will prove this theorem, and continue on to translate the other types of ITT into the computational setting. As an equivalent of the identity type, we will introduce an *equality* type $\mathsf{Eq}_A(M, M')$, which internalizes the equality judgment $M \doteq M' \in A$. We will interpret the formal $\mathsf{Id}_A(M, M')$ type using this equality type, erasing the computationally trivial identity eliminator in the process.

References

Thorsten Altenkirch, Conor McBride, and Wouter Swierstra. Observational equality, now! In *Proceedings of the ACM Workshop Programming Languages meets Program Verification, PLPV 2007, Freiburg, Germany, October 5, 2007*, pages 57–68, 2007.

Carlo Angiuli, Kuen-Bang Hou (Favonia), and Robert Harper. Computational higher type theory III: Univalent universes and exact equality. `https://arxiv.org/abs/1712.01800`, 2017.

Cyril Cohen, Thierry Coquand, Simon Huber, and Anders Mörtberg. Cubical type theory: a constructive interpretation of the univalence axiom. `https://arxiv.org/abs/1611.02108`, 2016.

Per Martin-Löf. An intuitionistic theory of types: predicative part. In H.E. Rose and J.C. Shepherdson, editors, *Logic Colloquium '73, Proceedings of the Logic Colloquium*, volume 80 of *Studies in Logic and the Foundations of Mathematics*, pages 73–118. North-Holland, 1975.

Per Martin-Löf. Constructive mathematics and computer programming. In *Logic, Methodology and Philosophy of Science VI, Proceedings of the Sixth International Congress of Logic, Methodology and Philosophy of Science, Hannover 1979*. 1982.

The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. `https://homotopytypetheory.org/book`, Institute for Advanced Study, 2013.