Computational Higher Type Theory (CHTT)

Robert Harper

Lecture Notes of Week 10 by Travis Hance and Paul Gölz

1 Properties of our type system

March 20, 2018

Last week, we formulated a mathematical account of the semantics of computational type theory. First, we defined an operator $\mathcal{T}$ which operates on "pre-type systems" or "candidate type systems". We used the Knaster-Tarski Theorem to find the least fixed point τ_0 such that $\mathcal{T}(\tau_0) \subseteq \tau$.

The candidate type system τ is a collection of triples (A, B, φ) where φ is a binary relation on closed values. A triple (A, B, φ) means that A and B are equal canonical types, with φ determining the member relation on values. Furthermore, we define

- $\tau^{\Downarrow}(A, B, \varphi)$ iff $A \Downarrow V$, $B \Downarrow W$, and $\tau(V, W, \varphi)$.
- $\varphi^{\Downarrow}(M, N)$ iff $M \Downarrow V$, $N \Downarrow W$, and $\varphi(V, W)$.

The $\tau^{\Downarrow}$ and $\varphi^{\Downarrow}$ are essentially τ and φ but where we allow the arguments to be terms instead of requiring them to be values.

The operator $\mathcal{T}$ is defined as

$$\mathcal{T}(\tau) = \textsc{bool}(\tau) \cup \textsc{pi}(\tau) \cup \textsc{sigma}(\tau) \cup \textsc{eq}(\tau) \cup \textsc{nat}(\tau).$$

$\mathcal{T}$ is monotone by construction.

We'll repeat a few rules here, but they can be found in last week's notes, Section 3.2.4.

- We define BOOL as

  $$\textsc{bool}(-) = \{(\mathsf{bool}, \mathsf{bool}, \beta)\} \text{ where } \beta = \{(\mathsf{true}, \mathsf{true}), (\mathsf{false}, \mathsf{false})\}.$$

- We define NAT as

  $$\textsc{nat}(-) = \{(\mathsf{Nat}, \mathsf{Nat}, \varphi_{\mathsf{Nat}})\}$$

  where φ_{Nat} is the least fixed point containing

  $$\{\textsc{zero}, \textsc{zero})\} \cup \{(\textsc{succ}(M), \textsc{succ}(N)) \mid \varphi^{\Downarrow}_{\mathsf{Nat}}(M, N)\}.$$

  Note that here we are using another induction inside the larger one. We can think of defining the Nat type as a "horizontal induction" and defining τ_0 as a "vertical induction."

- The equality type $\mathsf{Eq}_A(M, N)$ is defined in terms of a type A, so the definition of $\textsc{eq}(\tau)$ has to be defined in terms of τ.

  We say that $\textsc{eq}(\tau)(\mathsf{Eq}_A(M, N), \mathsf{Eq}_{A'}(M', N'), \varphi)$ is true iff

  - $\tau^{\Downarrow}(A, A', \varphi)$,
  - $\varphi^{\Downarrow}(M, M')$ and $\varphi^{\Downarrow}(N, N')$,

– $\varphi(\mathsf{refl}, \mathsf{refl})$ iff $\varphi^{\Downarrow}(M', N)$ and $\varphi^{\Downarrow}(M, N')$.

Essentially, this means that A and A' are equal types with membership described by φ, and M and M' are equal computations according to φ, and same for N and N'.

Now, τ_0 is a candidate type system (by construction) but we would like to prove that it is a real type system. To do this, we need to show it has the following properties:

1. **Functionality:** $\tau_0(A, B, \phi)$ and $\tau_0(A, B, \phi')$ imply $\phi = \phi'$
2. **P.E.R valued:** if $\tau_0(A, B, \phi)$, then ϕ is a partial equivalence relation
3. **Symmetry:** $\tau_0(A, B, \phi)$ implies $\tau_0(B, A, \phi)$
4. **Transitivity:** $\tau_0(A, B, \phi)$ and $\tau_0(B, C, \phi)$ imply $\tau_0(A, C, \phi)$.

To just give an idea of how these proofs we go, we will give a sketch of the proof of Functionality.

Theorem (Functionality)**.** *Let τ_0 be the least fixed point of $\mathcal{T}$. Then if $\tau_0(A, B, \phi)$ and $\tau_0(A, B, \phi')$, then $\phi = \phi'$.*

Proof. Given τ_0, define another type candidate τ as

$$\tau(A, B, \varphi) = \forall \varphi'.\ \tau_0(A, B, \varphi') \implies \varphi = \varphi').$$

Essentially, this is τ_0 but without the triples (A, B, φ) where φ is not unique for A and B.

We will show that $\mathcal{T}(\tau) \subseteq \tau$. This makes τ a fixed point, and since τ_0 is the *least* fixed point, we will have $\tau_0 \subseteq \tau$. Then we will have the desired property for τ_0 as well.

So let us show that $\mathcal{T}(\tau) \subseteq \tau$. Let $x \in \mathcal{T}(\tau)$. By definition of $\mathcal{T}$, we have either $x \in \textsc{bool}(\tau)$, $x \in \textsc{pi}(\tau)$, $x \in \textsc{sigma}(\tau)$, $x \in \textsc{eq}(\tau)$, or $x \in \textsc{nat}(\tau)$. We need to show that $x \in \tau$.

We will handle two of these cases here, just to illustrate how it goes: an easier case and a harder case.

- *Case $x \in \textsc{bool}(\tau)$.*

  In this case we must just have $x = (\mathsf{bool}, \mathsf{bool}, \beta)$ as defined above. By definition of τ, we have,

  $$\tau(\mathsf{bool}, \mathsf{bool}, \beta) = \forall \varphi'.\ \tau_0(\mathsf{bool}, \mathsf{bool}, \varphi') \implies \varphi' = \beta.$$

  But $\tau_0 = \mathcal{T}(\tau_0)$ and in the definition of $\mathcal{T}$ we can check that for $A = \mathsf{bool}$ and $B = \mathsf{bool}$ we must have $\varphi' = \beta$. Therefore, $x \in \tau$.

  This case was straightforward because $\textsc{bool}(\tau)$ does not actually depend on τ.

- *Case $x \in \textsc{pi}(\tau)$.*

  As a reminder, we have $\textsc{pi}(\tau)(A_0, A_0', \rho)$ iff

  – $\exists A, A', B, B'$ such that $A_0 = (x : A \to B)$ and $A_0' = (x : A' \to B')$,
  – $\exists \alpha$ such that $\tau(A, A', \alpha)$,
  – $\exists \varphi$ such that $\forall M, M'.\ \alpha^{\Downarrow}(M, M') \implies \tau^{\Downarrow}([M/x]B, [M'/x]B', \varphi(M, M'))$,
  – $\rho(M, M')$ iff
    * $\exists N.\ M = \lambda x.N$,
    * $\exists N'.\ M' = \lambda x.N'$,
    * $\forall P, P'.\ \alpha^{\Downarrow}(P, P') \implies \varphi(P, P')^{\Downarrow}([P/x]N, [P'/x]N')$.

Therefore, we have $x = (A_0, A_0', \rho)$ where $A, A', B, B, \alpha, \varphi$ all exist as described.

Furthermore, by definition of τ, we must have that α and φ are the *unique* binary relations satisfying these properties, that is,

- $\tau_0(A, A', \alpha') \implies \alpha = \alpha'$.
- $\forall M, M'.\ \alpha^{\Downarrow}(M, M') \implies \tau_0^{\Downarrow}([M/x]B, [M'/x]B', \gamma) \implies \varphi(M, M') = \gamma$.

Furthermore, given unique $A, A', B, B', \alpha, \varphi$, the definition uniquely defines ρ.

Therefore,

$$\tau_0(A_0, A_0', \rho') \implies \rho = \rho'.$$

Therefore, $(A_0, A_0', \rho) \in \tau$.

□

2 Equality and transport

Let's revisit Intensional Type Theory (ITT).

We inductively defined the judgments,

$$\begin{array}{ll} \Gamma \vdash M : A & \Gamma \vdash M \equiv M' : A \\ \Gamma \vdash A\ \mathsf{type} & \Gamma \vdash A \equiv A' \\ \Gamma\ \mathsf{ctxt} & \Gamma \equiv \Gamma' \end{array}$$

We also showed that if $\Gamma \vdash M : A$, then $|\Gamma| \gg |M| \in |A|$, where $|\cdot|$ denotes "erasure compilation", i.e., removing all type lambda application and abstraction from the expression. We also saw that $\mathsf{Id}_A(M, N)$ is not an adequate account of equality. On the plus side, it is a partial equivalence relation and it satisfies the indiscernibility of identicals (that is, if $\mathsf{Id}_A(M, N)$ then M can be substituted for N.

On the negative side, there is no notion of function extensionality. Id_A is defined uniformly over A, that is, the same way for each A. Therefore, we cannot possibly have $abs \equiv id : \mathsf{nat} \to \mathsf{nat}$, even though we would like to, because we must have $abs \not\equiv id : \mathsf{int} \to \mathsf{int}$.

Before we can address this, we need to motivate the notion of *universes*. Let's talk more about transport and the indiscernibility of identicals. We will show that if $\mathsf{Id}_A(M, N)$ is true then we can substitute M for N.

Theorem (Transport)**.** *Suppose B is a "predicate family".*

$$\Gamma, a : A \vdash B\ \mathit{type}.$$

Also, suppose that M is identical to N, with proof of identification P,

$$P : \mathsf{Id}_A(M, N),$$

and finally suppose that $[M/a]B$ is true,

$$Q : [M/a]B.$$

Then we can substitute N instead, that is we can find an element

$$\mathit{tr}[a.B](P)(Q) : [N/a]B.$$

Proof. We can take

$$\mathsf{tr}[a.B](P)(Q) = J_{a,b,\_.[a/a]B \to [b/a]B}(a.\lambda b : B.b)(Q).$$

□

Remark 1. *By β simplification, we can find that*

$$\mathsf{tr}[a.b](\mathsf{refl}_A(M))(Q) \equiv Q : [M/a]B.$$

Again, the only equivalence on which this works is definitional equivalence. We could try to define a funext object to handle function extensions; however, we wouldn't be able to say how tr or J works.

Now, we will try to use tr to solve a seemingly simple problem. We want to prove that $0 \neq 1$, or in other words, find M such that

$$\Gamma \vdash Q : \mathsf{Id}_{\mathsf{Nat}}(\mathsf{zero}, \mathsf{one}) \vdash M : \bot.$$

(Here, one is simply succ(zero).)

This should be easy; we just have to use Q to substitute zero for one. For any P, we do have,

$$\Gamma \vdash Q : \mathsf{tr}[a.P](Q) : [\mathsf{zero}/a]P \to [\mathsf{one}/a]P.$$

So we would be done if we could find P such that $[\mathsf{zero}/a]P \equiv \top$ and $[\mathsf{one}/a]P \equiv \bot$.

Could we simply define P as follow?

$$P = \mathsf{NATREC}(\top; \_.\_.\bot)(a).$$

We can't, simply because there is no such thing as NATREC in ITT!

In fact, one can show that there is no such P in ITT. This is a problem, and we would like to fix it. There are two possibilities.

1. Define a type theory with these "large elimination forms" (like NATREC) built in.
2. Allow us to just use natrec.

We would like to do option 2. But this requires us to ask, what *type* would the use of natrec have in $\mathsf{natrec}(\top; \_.\_.\bot)(a)$ have? To answer this, we need a "type of types," and this brings us to universes.

3 Universes

March 22, 2018

To prove that zero and one are not equal, we need to be able to define a type family $P(a)$ by induction on natural numbers, which is not possible in ITT.

We might attempt to change this by adding natural recursion on the type level as a type constructor NATREC:

$$\frac{\Gamma \vdash A_0\ \mathsf{type} \quad \Gamma, a : \mathsf{Nat}, b : \textcircled{?} \vdash A_1\ \mathsf{type} \quad \Gamma \vdash M : \mathsf{Nat}}{\Gamma \vdash \mathsf{NATREC}(A_0; a, b.\ A_1)(M)\ \mathsf{type}}$$

Even apart from the cumbersome repetition between definitions on the term and type level, it is not clear what "type" to give to b in the premise.

What we need is a type of types, a *universe* $\mathcal{U}$. Membership in this type implies being a type:

$$\frac{\Gamma \vdash A : \mathcal{U}}{\Gamma \vdash A\ \mathsf{type}}$$

Then, we can do away with the distinction between terms and types, and use the usual rule for natrec to type $P(a) \coloneqq \mathsf{natrec}(\top; \_, \_.\ \bot)(a)$:

$$\frac{\Gamma \vdash A_0 : \mathcal{U} \quad \Gamma, a : \mathsf{Nat}, b : \mathcal{U} \vdash A_1 : \mathcal{U} \quad \Gamma \vdash M : \mathsf{Nat}}{\Gamma \vdash \mathsf{natrec}(A_0; a, b.\ A_1)(M) : \mathcal{U}}$$

To show, for example, that $P(\mathsf{zero}) \equiv \top$, we lift the structural equality $P(\mathsf{zero}) \equiv \top : \mathcal{U}$ using the following rule:

$$\frac{\Gamma \vdash A \equiv B : \mathcal{U}}{\Gamma \vdash A \equiv B}$$

We then fill the universe with the types of ITT:

$$\Gamma \vdash \mathsf{Nat}, \mathsf{Bool}, \mathsf{Void}, \mathsf{Unit} : \mathcal{U} \qquad \frac{\Gamma \vdash A : \mathcal{U} \quad \Gamma, a : A \vdash B : \mathcal{U}}{\Gamma \vdash \Pi a : A.\ B, \Sigma a : A.\ B : \mathcal{U}}$$

$$\frac{\Gamma \vdash A : \mathcal{U} \quad \Gamma \vdash M, N : A}{\Gamma \vdash \mathsf{Id}_A(M, N) : \mathcal{U}}$$

And so forth.

Ideally, we would like $\mathcal{U}$ to encode all types. However, we cannot set $\mathcal{U} : \mathcal{U}$ because this would be inconsistent by Russel's paradox. To solve this problem, we define not just a single universe $\mathcal{U}$, but an infinite hierarchy of universes $\mathcal{U}_0 : \mathcal{U}_1 : \mathcal{U}_2 \ldots$. Every universe is maximal in the sense that it is closed under our type constructors by the aforem-entioned rules. This hierarchy is cumulative as per the following rule:

$$\frac{\Gamma \vdash M : \mathcal{U}_i}{\Gamma \vdash M : \mathcal{U}_{i+1}}$$

It is worth pointing out that our types can now include arbitrarily complicated computation.

4 Higher identification

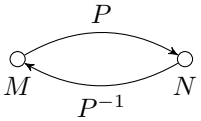
If Id_A should be a notion of equality, we would expect for example $\mathsf{Id}_{\mathsf{Nat}\to\mathsf{Nat}}(f, g)$ to be inhabited iff $\Pi a : \mathsf{Nat}.\ \mathsf{Id}_{\mathsf{Nat}}(f\ a, g\ a)$ is.

Now that we introduced universes, what should $\mathsf{Id}_{\mathcal{U}}$ look like? It would be natural to consider two types $A, B : \mathcal{U}$ as equal iff they are isomorphic, i.e., if there are functions $f : A \to B$ and $g : B \to A$ such that $g \circ f \ominus id$ and $f \circ g \ominus id$. But which relation should we use for "$\ominus$"? Requiring structural equality as $g(f(a)) \equiv a$ is way to fine. A better idea would be to require $\mathsf{Id}_A(g(f(a)), a)$, and Vladimir Voevodsky's univalence goes into that direction.

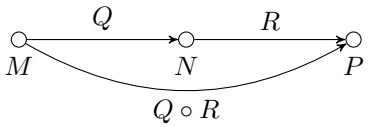
In section 2 of Week 7, we considered different ways of constructing elements of identity types:

- $\mathsf{refl}_A(M) : \mathsf{Id}_A(M, M)$. Leaving the type implicit, we suggestively set $id \coloneqq \mathsf{refl}_A(M)$. We illustrate this as follows:

- $\mathsf{sym}_A(P) : \mathsf{Id}_A(N, M)$ for $P : \mathsf{Id}_A(M, N)$. Set $P^{-1} \coloneqq \mathsf{sym}_A(P)$.



- $\mathsf{trans}_A(Q, R) : \mathsf{Id}_A(M, P)$ for $Q : \mathsf{Id}_A(M, N)$ and $R : \mathsf{Id}_A(N, P)$. Denote this by $Q \circ R$.



We would expect this structure to respect the groupoid[1] laws:

1. $id^{-1} \ominus id$
2. $Q \circ id \ominus Q$
3. $id \circ R \ominus R$
4. $P^{-1} \circ P \ominus id$
5. $P \circ P^{-1} \ominus id$
6. $P \circ (Q \circ R) \ominus (P \circ Q) \circ R$
7. $(P^{-1})^{-1} \ominus P$

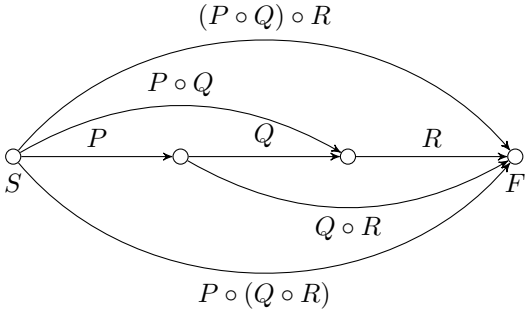
As we discussed in Week 7, choosing "$\equiv$" for "$\ominus$" only satisfies some of these laws. For example, property 1 is true, whereas at most one out of properties 2 and 3 holds, depending on the exact program used to implement transitivity.

To get the groupoid structure, we instead choose $\mathsf{Id}_{\mathsf{Id}_A}$ for "$\ominus$":

For instance, consider Law 5: Suppose that we have $P : \mathsf{Id}_A(M, N)$ and thus $P^{-1} : \mathsf{Id}_A(n, M)$. Then, by using $id^{-1} \equiv id$ and $id \circ id \equiv id$, one verifies that

$$J_{a,\_,p.\ \mathsf{Id}_{\mathsf{Id}_A(a,a)}(p \circ p^{-1}, \mathsf{refl}_A(a))}(a.\ \mathsf{refl}_{\mathsf{Id}_A(a,a)}(\mathsf{refl}_A(a)))(P) : \mathsf{Id}_{\mathsf{Id}_A(M,M)}(P \circ P^{-1}, \mathsf{refl}_A(M)).$$

Or consider Law 6:



Again, given P, Q, R, we can find a term of type $\mathsf{Id}_{\mathsf{Id}_A(S,F)}((P \circ Q) \circ R, P \circ (Q \circ R))$. Borrowing intuitions from homotopy theory, this term can be thought of as a continuous deformation (a *homotopy*) of the path corresponding to $(P \circ Q) \circ R$ into the path corresponding to $P \circ (Q \circ R)$.

Generally, the groupoid laws hold "up to higher identification". We call the nesting depth of $\mathsf{Id}_{\mathsf{Id}_{\ddots}}$ its dimension. Again, in the language of homotopy theory, this corresponds to going from points (dimension 0) to paths between points (dimension 1) to paths between paths between points (dimension 2) and so forth. Since the groupoid laws for objects of some dimension hold up to an equivalence of one higher dimension, the iterated Ids form a so-called ∞-*groupoid*.

[1] A groupoid is a generalization of a group in which elements are "typed." Here, we can only concatenate paths P and Q as $P \circ Q$ if $P : \mathsf{Id}_A(M, N)$ and $Q : \mathsf{Id}_A(N, P)$, i.e., if the endpoint of path P equals the starting point of path Q.

5 Outlook

When is $f : A \to B$ a bijection? One way of defining this is the following: For every element $b : B$, call its preimages $f^{-1}(b) \coloneqq \Sigma a : A.\ \mathsf{Id}_B(f(a), b)$ its *homotopy fiber*. f is a bijection iff every fiber contains exactly one element. Equivalently, we might require that every fiber contains a *center of contraction* c such that, for every element c' of the fiber, there is a "path from c to c'", i.e., a term of type $\mathsf{Id}_A(c, c')$. In this case, we say that the fibers are contractible.

Next week, we will study the univalence axiom, which is based on this idea.

References