Computational Higher Type Theory (CHTT)

Robert Harper

Lecture Notes of Week 8 by Yue Niu and Jonathan Laurent

1 Hypothetico-General Judgments

Last week, we looked briefly at the semantic judgment of computational dependent type theory with a one element context. This can be extended hypothetical-generally with the idea functionality:

Consider a h-g judgment:

$$x_1 : A_1, ..., x_n : A_n \gg A \doteq A' \ \mathsf{type}$$

and the corresponding judgment on terms:

$$x_1 : A_1, ..., x_n : A_n \gg M \doteq M' \in A$$

They are defined mutually recursively by induction on n:

$n = 0$: $\quad \cdot \gg A \doteq A'$ type iff $A \doteq A'$ type and
$\cdot \gg M \doteq M' \in A$ iff $M \doteq M' \in A$ (equivalent to the categorical judgments)

$n = k+1$: $\quad x_1 : A_1, ..., x_{k+1} : A_{k+1} \gg A \doteq A'$ type iff
$M_1 \doteq M_1' \in A_1$, $M_2 \doteq M_2' \in [M_1/x_1]A_2$,..., $M_{k+1} \doteq M_{k+1}' \in [M_i/x_i]_{i=1}^{k+1} A_{k+1}$ implies
$[M_i/x_i]_{i=1}^{k+1} A \doteq [M_i'/x_i]_{i=1}^{k+1} A'$ type and
$x_1 : A_1, ..., x_{k+1} : A_{k+1} \gg M \doteq M' \in A$ iff
$M_1 \doteq M_1' \in A_1$, $M_2 \doteq M_2' \in [M_1/x_1]A_2$,..., $M_{k+1} \doteq M_{k+1}' \in [M_i/x_i]_{i=1}^{k+1} A_{k+1}$ implies
$[M_i/x_i]_{i=1}^{k+1} M \doteq [M_i'/x_i]_{i=1}^{k+1} M' \in [M_i/x_i]_{i=1}^{k+1} A$

Note we define the unary versions of the h-g judgments as reflexive cases (e.g. $\Gamma \gg A$ type iff $\Gamma \gg A \doteq A$ type). We can check that the h-g judgments are symmetric and transitive, and by the "p.e.r" trick, we can get reflexivity as well. In addition, the h-g jugdments also have the structural properties of entailment:

1. $\Gamma, x : A, \Gamma' \gg x \in A$
2. if $\Gamma \gg \mathcal{J}$ and $\Gamma \gg A$ type then $\Gamma, x : A \gg \mathcal{J}$
3. if $\Gamma, x : A, \Gamma' \gg \mathcal{J}$ type and $\Gamma \gg M \doteq M' \in A$ then $\Gamma[M/x]\Gamma' \gg [M/x]\mathcal{J}$ type

Where $\mathcal{J}$ any antecedant of a semantic h-g judgment. With these properties, we can embed a logic in this framework:

1. inhabitantion: A true iff there is a term M s.t. $M \in A$
2. equality of proofs: $\mathsf{Eq}_A(M, N)$

1.1 Function Equality

Semantic function equality is extensional, since functionality is built into the definition of the semantic judgments:

$$\lambda x{:}A.\, M \doteq \lambda x{:}A.\, M' \in (x : A) \to B \text{ iff } x : A \gg M \doteq M' \in B$$

where the context is a mapping of variables to *closed* values.

1.2 Equality Type

The type $\mathsf{Eq}_A(M, N)$ internalizes semantic judgmental equality:

$$\begin{aligned}\mathsf{Eq}_A(M,N) \doteq \mathsf{Eq}_{A'}(M',N') \ \mathsf{type} \text{ iff } A \doteq A' \ \mathsf{type}, M \doteq M' \in A, \text{ and } N \doteq N' \in A\\ \mathsf{refl}_A(M) \in \mathsf{Eq}_A(M,N) \text{ iff } M \doteq N \in A\end{aligned}$$

2 FTT and CTT

As before, we relate FTT and CTT with theorems that state the sufficiency of the former for the latter. First, we extend the erasure operation to identity types, introduction, and elimination:

$$\begin{aligned}|\mathsf{Id}_A(M,N)| &\triangleq \mathsf{Eq}_{|A|}(|M|,|N|)\\ |\mathsf{refl}_A(M)| &\triangleq \underline{\mathsf{refl}}\\ |\mathsf{J}_{a,b,c.C}(a.Q)(P)| &\triangleq [|M|/a]|Q| \quad (\text{where } P : \mathsf{Id}_A(M,N))\end{aligned}$$

1. If $\Gamma \vdash M : A$ then $|\Gamma| \gg |M| \in |A|$
2. If $\Gamma \vdash M \equiv M' : A$ then $|\Gamma| \gg |M| \doteq |M'| \in |A|$
3. If $\Gamma \vdash A$ type then $|\Gamma| \gg |A|$ type
4. If $\Gamma \vdash A \equiv A'$ type then $|\Gamma| \gg |A| \doteq |A'|$ type

The proof proceeds by nested induction on the 4 FTT type derivations. We show the crucial case of typing for J.

Proof.

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

  By IH on the third premise, we have $|\Gamma| \gg |P| \in |\mathsf{Id}_A(M,N)|$. Unfolding the definitions of erasure and Eq, we have $|\Gamma| \gg |M| \doteq |N| \in |A|$. By IH on the second premise, we have $|\Gamma, a : A| \gg |Q| \in |[a, a, \mathsf{refl}_A(a)/a, b, p]C|$. Unfolding the erasure, we get $|\Gamma|, a : |A| \gg |Q| \in [a, a, \underline{\mathsf{refl}}/a, b, p]|C|$. By the substitution property, we get $|\Gamma| \gg [|M|/a]|Q| \in [|M|, |M|, \underline{\mathsf{refl}}/a, b, p]|C|$. Now by IH on the first premise, we have $|\Gamma, a : A, b : A, p : \mathsf{Id}_A(a, b)| \gg |C| \doteq |C|$ type, or that $|\Gamma|, a : |A|, b : |A|, p : \mathsf{Eq}_{|A|}(|a|, |b|) \gg |C| \doteq |C|$ type. By functionality on $|\Gamma| \gg |M| \doteq |M| \in |A|$, $|\Gamma| \gg |M| \doteq |N| \in |A|$, and $|\Gamma| \gg |P| \doteq \underline{\mathsf{refl}} \in \mathsf{Eq}_{|A|}(|M|, |N|)$, we have $|\Gamma| \gg [|M|, |M|, \underline{\mathsf{refl}}/a, b, p]|C| \doteq [|M|, |N|, |P|/a, b, p]|C|$ type. Thus we get $|\Gamma| \gg [|M|/a]|Q| \in [|M|, |N|, |P|/a, b, p]|C|$, which by definition is $|\Gamma| \gg |\mathsf{J}_{a,b,c.C}(a.Q)(P)| \in |[M, N, P/a, b, p]C|$.

□

3 An Explicit Construction of Computational Dependent Types

In this section, we give a rigorous set-theoretic construction of computational dependent types (CDTs), as introduced by Allen Allen [1987]. In this construction, we define the system of computational dependent types as the least fixed point of a monotone operator over a lattice of candidate type-systems. The existence of such a fixed point is guaranteed by the Knaster-Tarski theorem.

3.1 Background: Knaster-Tarski Fixed Point Theorem

A *complete lattice* $(L, \leq)$ consists in a set L along with a pre-order[1] $\leq$ such that all subsets of L admit a greatest lower bound (aka *meet*) and a least upper bound (aka *join*). More formally, for every subset $X \subseteq L$, there are elements $\bigwedge X$ and $\bigvee X$ in L such that:

$$\forall x \in X,\ \bigwedge X \leq x \quad \text{and} \quad \forall y, (\forall x \in X,\ y \leq x) \implies y \leq \bigwedge X$$

$$\forall x \in X,\ x \leq \bigvee X \quad \text{and} \quad \forall y, (\forall x \in X,\ x \leq y) \implies \bigvee X \leq y.$$

Remark. *If L is infinite, this is a stronger requirement than just having binary meet and join operators $\wedge$ and $\vee$ on L along with smallest and largest elements $\top$ and $\bot$ (in which case $(L, \leq)$ is called a* plain lattice*).*

A function $f : L \to L$ is said to be *monotone* if it preserves $\leq$ in the sense that

$$\forall x, y \in L,\ x \leq y \implies f(x) \leq f(y).$$

We can now state Knaster-Tarski theorem.

Theorem. *Let $(L, \leq)$ a complete lattice and $f : L \to L$ a monotone function. Then, f admits a complete lattice of fixed points. In particular, it admits a least fixed point μf which is given by the meet of its pre-fixed points:*

$$\mu f = \bigwedge \{x \in L \mid f(x) \leq x\}.$$

In practice, $(L, \leq)$ is often a powerset lattice: $(L, \leq) = (\mathcal{P}(E), \subseteq)$ where E is a set and $\subseteq$ refers to set inclusion. In this case, the meet operation corresponds to set intersection and the join operation corresponds to set union. Therefore, if $f : \mathcal{P}(E) \to \mathcal{P}(E)$ is monotone, μf is the intersection of every subset of E that is *closed under* f:

$$\mu f = \bigcap \{X \subseteq E \mid f(X) \subseteq X\}.$$

[1]A pre-order is a reflexive and transitive relation.

3.2 Construction of Computational Dependent Types

3.2.1 Candidate type systems

A *candidate type-system* is a ternary relation between values, values, and binary relations over values. Intuitively, if $\tau(A, B, \phi)$ then A and B are closed values corresponding to equal types and ϕ is a binary relation over closed values that corresponds to equality of values within type A (or B). For example, a type system for booleans would be

$$\tau_{\mathsf{bool}} = \{(\mathsf{bool}, \mathsf{bool}, \beta)\}, \qquad \beta = \{(\mathsf{true}, \mathsf{true}), (\mathsf{false}, \mathsf{false})\}.$$

3.2.2 Real type systems

A *real type-system* is defined as a candidate type system for which the following properties hold (every free variable is universally quantified):

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

Remark. *Another way to formulate transitivity would be as follows:*

$$\tau(A, B, \phi) \textit{ and } \tau(B, C, \phi') \textit{ imply } \tau(A, C, \phi) \textit{ and } \phi = \phi'.$$

This is equivalent to our first formulation. Indeed, suppose $\tau(A, B, \phi)$ and $\tau(B, C, \phi')$. Using symmetry and transitivity, we have $\tau(B, B, \phi)$ and $\tau(B, B, \phi')$ (p.e.r trick). Using functionality, we get $\phi = \phi'$.

The set of all candidate type-systems forms a complete lattice for set inclusion. This is **not** true of the set of all real type-systems, which is why we need the notion of candidate type-system (see section 3.2.4).

3.2.3 Truth judgments relative to a real type system

We define the following truth judgments relative to a real type-system τ:

- $\tau \models A \doteq B$ type if and only if $\tau^{\Downarrow}(A, B, \phi)$ for some ϕ
- $\tau \models M \doteq N \in A$ if and only if $\tau^{\Downarrow}(A, B, \phi)$ and $\phi^{\Downarrow}(M, N)$ for some ϕ

where the lifted relations $\tau^{\Downarrow}$ and $\phi^{\Downarrow}$ are defined as follows:

- $\phi^{\Downarrow}(M, N)$ iff there exists values V, W such that $M \Downarrow V$, $N \Downarrow W$ and $\phi(V, W)$
- $\tau^{\Downarrow}(A, B, \phi)$ iff. there exists values V, W such that $A \Downarrow V$, $B \Downarrow W$ and $\tau(V, W, \phi)$.

[2]A partial equivalence relation is a symmetric and transitive relation.

3.2.4 Defining τ_0 as a least fixed point

We can define the type system τ_0 of computational dependent types as the least fixed-point of a monotone operator

$$\Phi(\tau) \triangleq \textsc{bool} \cup \textsc{nat} \cup \textsc{eq}(\tau) \cup \textsc{pi}(\tau) \cup \textsc{sigma}(\tau).$$

After we define BOOL, NAT, EQ, PI and TAU, we must prove that the resulting operator Φ is monotone and that $\tau_0 \triangleq \mu\Phi$ is a real type-system.

Booleans As seen earlier,

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

Natural numbers We define $\textsc{nat} = \{(\mathsf{nat}, \mathsf{nat}, \mu N)\}$ where μN is the least fixed-point of the following monotone operator over the complete lattice of relations over values:

$$N(\alpha) = \{(\mathsf{z}, \mathsf{z})\} \cup \{(\mathsf{s}(n), \mathsf{s}(m)) \mid \alpha(m, n)\}.$$

Equality We have $\textsc{eq}(\tau)(A_0, A'_0, \epsilon)$ if and only if

- $A_0 = \mathsf{Eq}_A(M, N)$ and $A'_0 = \mathsf{Eq}_{A'}(M', N')$ for some A, A', M, M', N, N'
- $\tau^{\Downarrow}(A, A', \phi)$, $\phi^{\Downarrow}(M, M')$ and $\phi^{\Downarrow}(N, N')$ for some ϕ
- $\epsilon = \{(\mathsf{refl}, \mathsf{refl})\}$ if $\phi^{\Downarrow}(M, N)$ and $\{\}$ otherwise.

Dependent function We have $\textsc{pi}(\tau)(A_0, A'_0, \rho)$ if and only if

- $A_0 = (x : A \to B)$ and $A'_0 = (x : A' \to B')$ for some A, A', B, B'
- $\tau(A, A', \alpha)$ for some α
- there exists a function ψ mapping pairs of terms to relations over values such that

  $$\alpha^{\Downarrow}(M, M') \text{ implies } \tau^{\Downarrow}([M/x]B,\ [M'/x]B',\ \psi(M, M')) \text{ for all terms } M, M'$$

- $\rho(M, M')$ if and only if
  - $M = \lambda x.N$ for some N
  - $M' = \lambda x.N'$ for some N'
  - $\alpha^{\Downarrow}(P, P')$ implies $\psi(P, P')^{\Downarrow}([P/x]N,\ [P'/x]N')$ for all terms P, P'.

Dependent products We have $\textsc{sigma}(\tau)(A_0, A'_0, \sigma)$ if and only if

- $A_0 = (x : A \times B)$ and $A'_0 = (x : A' \times B')$ for some A, A', B, B'
- $\tau(A, A', \alpha)$ for some α
- there exists a function ψ mapping pairs of terms to relations over values such that

  $$\alpha^{\Downarrow}(M, M') \text{ implies } \tau^{\Downarrow}([M/x]B,\ [M'/x]B',\ \psi(M, M')) \text{ for all terms } M, M'$$

- $\sigma(P, P')$ if and only if
  - $P = \mathsf{pair}_{A,x.B}(M, N)$ for some M, N
  - $P' = \mathsf{pair}_{A',x.B'}(M', N')$ for some M', N'
  - $\alpha^{\Downarrow}(M, M')$ and $\psi(M, M')^{\Downarrow}(N, N')$.

3.2.5 Proving the typing rules for τ_0

Finally, we must show that all the typical typing rules are true with respect to τ_0. For example, we must show that if $\tau_0 \models M : (x : A \to B)$ and $\tau_0 \models M : A$ then $\tau_0 \models MN : [N/x]B$.

3.3 Discussion

In order to show that Φ (as defined in 3.2.4) is monotone, it is sufficient to show that EQ, PI and SIGMA are monotone. A full proof would be tedious but the core idea that makes it work is that τ only appears in positive positions in the definitions of EQ(τ), PI(τ) and SIGMA(τ). In the case of PI for example, this reflects the fact that the terms A and $[M/x]B$ for all M can be considered as defined prior to $x : A \to B$.

This would not hold for the type constructor $\forall X.A$, which cannot regard $[B/X]A$ as prior to $\forall X.A$ for all B, as B could be $\forall X.A$. More concretely, here is an attempt of adding universal quantification over types in our language by extending Φ with a FORALL operator:

Type quantification We have FORALL$(\tau)(A_0, A_0', F)$ if and only if

- $A_0 = \forall X.A$ and $A_0' = \forall X.A'$
- There is a mapping ψ from pairs of terms to relations over values such that

$$\tau^{\Downarrow}(T, T', -) \text{ implies } \tau^{\Downarrow}([T/x]A, [T'/x']A', \psi(T, T')) \text{ for all terms } T, T'$$

- $F(M, M')$ if and only if
  - $M = \Lambda X.N$ for some term N
  - $M' = \Lambda X.N'$ for some term N'
  - $\tau^{\Downarrow}(T, T', -)$ implies $\psi(T, T')^{\Downarrow}([T/x]N, [T'/x]N')$ for all terms T, T'

Such an operator is **not** monotone though, as can be seen by the negative occurences of τ in the definition of FORALL(τ) (highlighted in red).

Similarly, in the definition of NAT, it is crucial that the N relation is monotone so that μN is well-defined. This monotonicity constraint prevents us from giving a fixed point definition for a type like $D \simeq D \to D$. Indeed, it is tempting to define a type-system for D as follows: $\tau_D \triangleq \{(D, D, \mu X)\}$ where μX is the least-fixed point of X:

$$X(\alpha) \triangleq \{(\mathsf{Fun}(f), \mathsf{Fun}(g)) \mid \forall V, W.\ \alpha(V, W) \implies \alpha^{\Downarrow}(f(V), f(W))\}$$

Such an operator is **not** monotone though, α appearing in a negative position in the definition of $X(\alpha)$. Therefore, μX does not necessarily exist. This makes sense because were a type like D allowed, we would have non-termination, which we do not want at this stage.

References

Stuart Allen. A non-type-theoretic definition of Martin-Löf's types. Technical report, Cornell University, 1987.