Partiality, State and Dependent Types

Kasper Svendsen[1], Lars Birkedal[1], and Aleksandar Nanevski[2]

[1] IT University of Copenhagen {kasv,birkedal}@itu.dk
[2] IMDEA Software aleks.nanevski@imdea.org

Abstract. Partial type theories allow reasoning about recursively-defined computations using fixed-point induction. However, fixed-point induction is only sound for admissible types and not all types are admissible in sufficiently expressive dependent type theories.

Previous solutions have either introduced explicit admissibility conditions on the use of fixed points, or limited the underlying type theory. In this paper we propose a third approach, which supports Hoare-style partial correctness reasoning, without admissibility conditions, but at a tradeoff that one cannot reason equationally about effectful computations. The resulting system is still quite expressive and useful in practice, which we confirm by an implementation as an extension of Coq.

1 Introduction

Dependent type theories such as the Calculus of Inductive Constructions [2] provide powerful languages for integrated programming, specification, and verification. However, to maintain soundness, they typically require all computations to be pure and terminating, severely limiting their use as general purpose programming languages.

Constable and Smith [9] proposed adding partiality by introducing a type $\bigcirc\tau$ of potentially non-terminating computations of type τ, along with the following fixed point principle for typing recursively defined computations:

$$\text{if } M : \bigcirc\tau \to \bigcirc\tau \quad \text{then} \quad \boldsymbol{fix}(M) : \bigcirc\tau$$

Unfortunately, in sufficiently expressive dependent type theories, there exists types τ for which the above fixed point principle is unsound [10]. For instance, in type theories with subset-types, the fixed point principle allows reasoning by a form of fixed point induction, which is only sound for admissible predicates (a predicate is admissible if it holds for the limit whenever it holds for all finite approximations). Previous type theories based on the idea of partial types which admit fixed points have approached the admissibility issue in roughly two different ways:

1. The notion of admissibility is axiomatized in the type theory and explicit admissibility conditions are required in order to use ***fix***. This approach has, e.g., been investigated by Crary in the context of Nuprl [10]. The resulting type theory is expressive, but admissibility conditions lead to significant proof obligations, in particular, when using Σ types.

2. The underlying dependent type theory is restricted in such a way that one can only form types that are trivially admissible. This approach has, e.g., been explored in recent work on Hoare Type Theory (HTT) [21]. The restrictions exclude usage of subset types and Σ types, which are often used for expressing properties of computations and for modularity. Another problem with this approach is that since it limits the underlying dependent type theory one cannot easily implement it as a simple extension of existing implementations.

In this paper we explore a third approach, which ensures that all types are admissible, not by limiting the underlying standard dependent type theory, but by limiting only the partial types. The limitation on partial types consists of equating all effectful computations at a given type: if M and N are both of type $\bigcirc\tau$, then they are propositionally equal. Thus, with this approach, the only way to reason about effectful computations is through their type, rather than via equality or predicates. With sufficiently expressive types, the type of an effectful computation can serve as a partial correctness specification of the computation. Our hypothesis is that this approach allows us to restrict attention to a subset of admissible types, which is closed under the standard dependent type formers and which suffices for reasoning about partial correctness.

To demonstrate that this approach scales to expressive type theories and to effects beyond partiality, we extend the Calculus of Inductive Constructions (CIC) [2] with stateful and potentially non-terminating computations. Since reasoning about these effectful computations is limited to their type, our partial types are further refined into a Hoare-style partial correctness specifications, and have the form $\boldsymbol{ST}\,\tau\,(P, Q)$, standing for computations with pre-condition P, post-condition Q, that diverge or terminate with a value of type τ.

The resulting type theory is an impredicative variant of Hoare Type Theory [17], which differs from previous work on Hoare Type Theory in the scope of features considered and the semantic approach. In particular, this paper is the first to clarify semantically the issue of admissibility in Hoare Type Theory.

Impredicative Hoare Type Theory (iHTT) features the universes of propositions (***prop***), small types (***set***), and large types (***type***), with ***prop*** included in ***set***, ***set*** included in ***type***, and axioms ***prop***:***type*** and ***set***:***type***. The ***prop*** and ***set*** universes are impredicative, while ***type*** is predicative. There are two main challenges in building a model to justify the soundness of iHTT: (1) achieving that Hoare types are small ($\boldsymbol{ST}\ \tau\ s : \boldsymbol{set}$), which enables *higher-order store*; that is, storing side-effectful computations into the heap, and (2) supporting arbitrary Σ types, and more generally, inductive types. In this respect iHTT differs from the previous work on Hoare Type Theory, which either lacks higher-order store [19], lacks strong Σ types [21], or whose soundness has been justified using specific syntactic methods that do not scale to fully general inductive definitions [17, 18].

The model is based on a standard realizability model of partial equivalence relations (PERs) and assemblies over a combinatory algebra A. These give rise to a model of the Calculus of Constructions [14], with ***set*** modelled using PERs. Restricting PERs to complete PERs (i.e., PERs closed under limits of ω-chains) over a suitable universal domain, allows one to model recursion in a simply-typed setting [4], or in a dependently-typed setting, but without strong Σ types [21].

Our contribution is in identifying a set of complete *monotone* PERs that are closed under Σ types and Hoare types. Complete PERs do not model Σ types because, given a chain of dependent pairs, in general, due to dependency, the second components of the chain are elements of *distinct* complete PERs. To apply completeness, we need a fixed single complete PER. Monotonicity will equate the first components of the chain and give us the needed single complete PER for the second components. Monotonicity further forces a trivial equality on Hoare types, equating all effectful computations satisfying a given specification. However, it does not influence the equality on the total, purely functional, fragment of iHTT, ensuring that we still model CIC. This is sufficient for very expressive Hoare-style reasoning, and avoids admissibility conditions on the use of ***fix***.

As iHTT is an extension of CIC, we have implemented iHTT as an axiomatic extension of Coq [1]. The implementation is carried out in Ssreflect [12] (a recent extension of Coq), based on the previous implementation of predicative Hoare Type Theory [19]. The implementation is available at:

`http://www.itu.dk/people/kasv/ihtt.tgz`.

2 Hoare types by example

To illustrate Hoare types, we sketch a specification of a library for arrays in iHTT. We assume that array indexes range over a finite type $\iota{:}\boldsymbol{set}_{fin}$, that the elements of ι can be enumerated as $\iota_0, \iota_1, \ldots, \iota_n$, and that equality between these elements can be decided by a function $=\!= : \iota \to \iota \to \boldsymbol{bool}$.

Each array is implemented as a contiguous block of locations, each location storing a value from the range type $\tau{:}\boldsymbol{set}$. The space occupied by the array is uniquely determined by ι, τ, and the pointer to the first element, justifying that the ***array*** type be defined as this first pointer.

$$\boldsymbol{array} : \boldsymbol{set}_{fin} \to \boldsymbol{set} \to \boldsymbol{set} = \lambda\iota.\, \lambda\tau.\, \boldsymbol{ptr}.$$

Here, ***ptr*** is the type of pointers, which we assume isomorphic to ***nat***. Each array is essentially a stateful implementation of some finite function $f{:}\iota{\to}\tau$. To capture this, we define a predicate indexed by f, that describes the layout of an array in the heap.

$$\begin{aligned}&\mathsf{shape} : (\boldsymbol{array}\ \iota\ \tau) \to (\iota \to \tau) \to \boldsymbol{heap} \to \boldsymbol{prop} = \\ &\quad \lambda a.\, \lambda f.\, \lambda h.\, h = a \mapsto f\ \iota_o \bullet a{+}1 \mapsto f\ \iota_1 \bullet \cdots \bullet a{+}n \mapsto f\ \iota_n.\end{aligned}$$

In other words, h stores an array a, representing a finite function f, if $\mathsf{shape}\ a\ f\ h$ holds, that is, if h consists of $n{+}1$ consecutive locations a, $a{+}1, \ldots, a{+}n$, storing $f\,\iota_0, f\,\iota_1, \ldots, f\,\iota_n$, respectively. The property is stated in terms of singleton heaps $a{+}k \mapsto f\,\iota_k$, connected by the operator $\bullet$ for *disjoint* heap union. Later in the text, we will also require a constant empty denoting the empty heap.

The type of arrays comes equipped with several methods for accessing and manipulating the array elements. For example, the method for reading the value at index $k{:}\iota$ can be given the following type.

$$\begin{aligned}\mathsf{read} : \mathit{\Pi} a{:}\boldsymbol{array}\ \iota\ \tau.\, \mathit{\Pi} k{:}\iota.&\\ \boldsymbol{ST}\ \tau\ (&\lambda h.\, \exists f.\, \mathsf{shape}\ a\ f\ h,\\ &\lambda r.\, \lambda h.\, \lambda m.\, \forall f.\, \mathsf{shape}\ a\ f\ h \to r = f\,k \wedge m = h)\end{aligned}$$

Informally, $\mathsf{read}\ a\ k$ is specified as a stateful computation whose precondition permits the execution only in a heap h which stores a valid array at adress a ($\exists f.\, \mathsf{shape}\ a\ f\ h$). The postcondition, on the other hand, specifies the result of executing $\mathsf{read}\ a\ k$ as a relation between output result $r{:}\tau$, input heap h and output heap m. In particular, the result r is indeed the array value at index k ($r = f\,k$), and the input heap is unchanged ($m = h$).

Unlike in ordinary Hoare logic, but similar to VDM [6], our postcondition is parametrized wrt. both input and the output heaps in order to directly express the relationship between the two. In particular, when this relationship depends on some specification-level value, such as f above, the dependency can be expressed by an ordinary propositional quantification.

Hoare types employ *small footprint* specifications, as in separation logic [20], whereby the specifications only describe the parts of the heap that the computation traverses. The untraversed parts are by default invariant. To illustrate, consider the type for the method new that generates a fresh array, indexed by ι, and populated by the value $x{:}\tau$.

$$\mathsf{new} : \mathit{\Pi} x{:}\tau.\, \boldsymbol{ST}\ (\boldsymbol{array}\ \iota\ \tau)\ (\lambda h.\, h = \mathsf{empty}, \lambda a.\, \lambda h.\, \lambda m.\, \mathsf{shape}\ a\ (\lambda z.\, x)\ m)$$

The type states *not* that $\mathsf{new}\ x$ can only run in an empty heap, but that $\mathsf{new}\ x$ *changes* the empty subheap of the current heap into a heap m containing an array rooted at a and storing all x's. In other words, new is *adding* fresh pointers, and the resulting array a itself is fresh. On the other hand, unlike in separation logic, we allow that the specifications can directly use and quantify over variables of type ***heap***. For completeness, we next simply list without discussion the types of the other methods for arrays.

$$\begin{aligned}&\mathsf{new\_from\_fun} : \mathit{\Pi} f{:}\iota{\to}\tau.\, \boldsymbol{ST}\ (\boldsymbol{array}\ \iota\ \tau)\ (\lambda h.\, h = \mathsf{empty}, \lambda a\,h\,m.\, \mathsf{shape}\ a\ f\ m)\\ &\mathsf{free} : \mathit{\Pi} a{:}\boldsymbol{array}\ \iota\ \tau.\, \boldsymbol{ST}\ \boldsymbol{unit}\ (\lambda h.\, \exists f.\, \mathsf{shape}\ a\ f\ h, \lambda r\,h\,m.\, m = \mathsf{empty})\\ &\mathsf{write} : \mathit{\Pi} a{:}\boldsymbol{array}\ \iota\ \tau.\, \mathit{\Pi} k{:}\iota.\, \mathit{\Pi} x{:}\tau.\\ &\qquad \boldsymbol{ST}\ \boldsymbol{unit}\ (\lambda h.\, \exists f.\, \mathsf{shape}\ a\ f\ h, \lambda r\,h\,m.\, \forall f.\, \mathsf{shape}\ a\ f\ h \to\\ &\qquad\qquad\qquad\qquad \mathsf{shape}\ a\ (\lambda z.\, \mathsf{if}\ z == k\ \mathsf{then}\ x\ \mathsf{else}\ f(z))\ m)\end{aligned}$$

At this point, we emphasize that various type theoretic abstractions are quite essential for practical work with Hoare types. The usefulness of $\mathit{\Pi}$ types and the propositional quantifiers is apparent from the specification of the array methods. But the ability to structure specification is important too. For example, we can pair pre- and postconditions into a type $\boldsymbol{spec}\ \tau = (\boldsymbol{heap} \to \boldsymbol{prop}) \times (\tau \to \boldsymbol{heap} \to \boldsymbol{heap} \to \boldsymbol{prop})$, which is then used to specify the fixpoint combinator.

$$\begin{aligned}\mathsf{fix} : &\ \mathit{\Pi} \alpha{:}\boldsymbol{set}.\, \mathit{\Pi} \beta{:}\alpha{\to}\boldsymbol{set}.\, \mathit{\Pi} s{:}\mathit{\Pi} x.\, \boldsymbol{spec}\ (\beta\ x).\\ &(\mathit{\Pi} x.\, \boldsymbol{ST}\ (\beta\ x)\ (s\ x) \to \mathit{\Pi} x.\, \boldsymbol{ST}\ (\beta\ x)\ (s\ x)) \to \mathit{\Pi} x.\, \boldsymbol{ST}\ (\beta\ x)\ (s\ x).\end{aligned}$$

Structuring proofs and specifications with programs is also necessary, and is achieved using dependent records (i.e., $\mathit{\Sigma}$ types), which we illustrate next.

The first example of a dependent record is the $\boldsymbol{set}_{fin}$ type. This is an algebraic structure containing the carrier type σ, the operation $==$ for deciding equality on σ, and a list enumerating σ's elements. Additionally, $\boldsymbol{set}_{fin}$ needs proofs that $==$ indeed decides equality, and that the enumeration list contains each element exactly once. Using the record notation $[x_1{:}\tau_1, \dots x_n{:}\tau_n]$ instead of the more cumbersome $\Sigma x_1{:}\tau_1 \dots \Sigma x_n{:}\tau_n.\, 1$, the $\boldsymbol{set}_{fin}$ type is defined as follows.

$$\begin{aligned}\boldsymbol{set}_{fin} = [\,&\sigma : \boldsymbol{set}, \mathsf{enum} : \boldsymbol{list}\ \sigma, \mathsf{==} : \sigma \to \sigma \to \mathsf{bool},\\ &\mathsf{eqp} : \forall x\, y{:}\sigma.\, x \mathsf{==} y = \mathsf{true} \iff x = y,\\ &\mathsf{enump} : \forall x{:}\sigma.\, \mathsf{count}\ x\ \mathsf{enum} = 1]\end{aligned}$$

The above dependent record refines a type. In practice, we will also use records that refine *values*. For example, in programming, arrays are often indexed by the type of bounded integers $I_n = [x : \boldsymbol{nat}, \mathsf{boundp} : x \leq n]$. I_n can be extended with appropriate fields, to satisfy the specification for $\boldsymbol{set}_{fin}$, but the important point here is that the elements of I_n are dependent records containing a number x and a proof that $x \leq n$. Of course, during actual execution, this proof can be ignored (proofs are computationally irrelevant), but it is clearly important statically, during verification.

Finally, the library of arrays itself can be ascribed a signature which will serve as an interface to the client programs. This signature too is a dependent record, providing types for all the array methods. Just as in the case of $\boldsymbol{set}_{fin}$ and I_n, the signature may also include properties, similar to object invariants [13, 15]. For example, we have found it useful in practice to hide from the clients the definitions of the array type and the array shape predicate, but expose that two arrays in stable states; that is, between two method calls, stored in compatible heaps must be equal (i.e., that the shape predicate is "functional"):

$$\begin{aligned}\mathsf{functional} : \Pi \mathsf{shape}.\, &\forall a_1\ a_2\ f_1\ f_2\ h_1\ h_2.\, \mathsf{shape}\ a_1\ f_1\ h_1 \to \mathsf{shape}\ a_2\ f_2\ h_2 \to\\ &(\exists j_1\ j_2.\, h_1 \bullet j_1 = h_2 \bullet j_2) \to a_1 = a_2 \wedge f_1 = f_2 \wedge h_1 = h_2.\end{aligned}$$

Then the signature for arrays indexed by ι, containing values of type τ, is provided by the following dependent record parametrized by ι and τ.

$$\begin{aligned}ArraySig = \Pi \iota{:}\boldsymbol{set}_{fin}.\, \Pi \tau{:}\boldsymbol{set}.\\ &[\,\boldsymbol{array} : \boldsymbol{set},\ \mathsf{shape} : \boldsymbol{array} \to (\iota \to \tau) \to \boldsymbol{heap} \to \boldsymbol{prop},\\ &\ \mathsf{funcp} : \mathsf{functional\ shape},\ \mathsf{read} : \Pi a{:}\boldsymbol{array}.\, \Pi k{:}\iota.\ \ \dots].\end{aligned}$$

Therefore, Σ types are central for building verified libraries of programs, specifications and proofs.

3 Semantics

In this section we present a model for impredicative Hoare Type Theory. We will introduce the relevant parts of iHTT as the model is defined. The full type theory is defined in Appendix A. In presenting the model, we first focus on the $\boldsymbol{set}$-universe, and then scale up to cover all of iHTT.

Since the purely-function fragment of IHTT is terminating, we take our universe of realizers to be a universal *pre-domain* with a suitable sub-domain for modelling stateful and potentially non-terminating computations.

Definition 1. *Let $\mathbb{V}$ denote a pre-domain satisfying the following recursive pre-domain equation:*

$$\mathbb{V} \cong 1 + \mathbb{N} + (\mathbb{V} \times \mathbb{V}) + (\mathbb{V} \to_c \mathbb{V}_\perp) + T(\mathbb{V}) + H(\mathbb{V})$$

where $\to_c$ is the space of continuous functions and

$$\begin{aligned}T(\mathbb{V}) &\overset{def}{=} H(\mathbb{V}) \to_c ((\mathbb{V} \times H(\mathbb{V})) + 1)_\perp\\ H(\mathbb{V}) &\overset{def}{=} \{h : \mathbf{ptr} \to_c \mathbb{V}_\perp \mid supp(f)\ finite \wedge f(\mathbf{null}) = \perp\}\\ supp(h : \mathbf{ptr} \to \mathbb{V}_\perp) &\overset{def}{=} \{l \in \mathbf{ptr} \mid f(l) \neq \perp\}\end{aligned}$$

The first four summands of $\mathbb{V}$ model the underlying dependent type theory, and $T(\mathbb{V})$ and $H(\mathbb{V})$ model computations and heaps, respectively. The ordering on $T(\mathbb{V})$ is the standard pointwise order and the ordering on $H(\mathbb{V})$ is as follows:

$$h_1 \leq h_2 \quad \textit{iff} \quad \textit{supp}(h_1) = \textit{supp}(h_2) \wedge \forall n \in \textit{supp}(h_1).\ h_1(n) \leq h_2(n)$$

Let $in_1, in_{\mathbb{N}}, in_{\times}, in_{\rightarrow}, in_T$, and in_H denote injections into $\mathbb{V}$ corresponding to each of the above summands.

$\mathbb{V}$ defines a PCA with the following partial application operator:

Definition 2. *Let $\cdot : \mathbb{V} \times \mathbb{V} \rightharpoonup \mathbb{V}$ denote the function,*

$$a \cdot b = \begin{cases} f(b) & \textit{if } a = in_{\rightarrow}(f) \wedge f(b) \neq \bot \\ \textbf{undef} & \textit{otherwise} \end{cases}$$

We recall some notation and definitions. If $R \subseteq A \times A$ is a PER then its domain, denoted $|R|$, is $\{x \in A \mid (x, x) \in R\}$. If $R, S \subseteq A \times A$ are PERs, then $R \rightarrow S$ is the PER $\{(\alpha, \beta) \in A \times A \mid \forall x, y \in A.\ (x, y) \in R \Rightarrow (\alpha \cdot x, \beta \cdot y) \in S\}$. If $R \subseteq A \times A$ is a PER and $f : A \rightarrow B$ then $f(R)$ denotes the PER $\{(f(x), f(y)) \mid (x, y) \in R\} \subseteq B \times B$. For a subset $X \subseteq A$, we use $\Delta(X)$ to denote the PER $\{(x, y) \mid x \in X \wedge y \in X\}$. Lastly, if $R \subseteq A \times A$ is a PER, we use $[R]$ to denote the set of R equivalence classes.

Definition 3 ($\mathrm{Per}(A)$). *The category of PERs, $\mathrm{Per}(A)$, over a partial combinatory algebra $(A, \cdot)$, has PERs over A as objects. Morphisms from R to S are set-theoretic functions $f : [R] \rightarrow [S]$, such that there exists a realizer $\alpha \in A$ such that,*

$$\forall e \in |R|.\ [\alpha \cdot e]_S = f([e]_R)$$

$\mathrm{Per}(\mathbb{V})$ is cartesian closed and thus models simple type theory. To model recursion, note that a realized set-theoretic function is completely determined by its realizers (i.e., $\mathrm{Per}(\mathbb{V})(R, S) \cong [R \rightarrow S]$) and that we have the standard least fixed-point operator on the sub-domain of computations of $\mathbb{V}$. This lifts to a least fixed-point operator on those PERs that are admissible on the sub-domain of computations:

Definition 4.

1. *A PER $R \subseteq A \times A$ on a pre-domain A is* complete *if, for all chains $(c_i)_{i \in \mathrm{N}}$ and $(d_i)_{i \in \mathrm{N}}$ such that $(c_i, d_i) \in R$ for all i, also $(\sqcup_i c_i, \sqcup_i d_i) \in R$.*
2. *A PER $R \subseteq A \times A$ on a domain A is* admissible *if is complete and $\bot \in |R|$.*

Let $\mathrm{CPer}(A)$ and $\mathrm{AdmPer}(A)$ denote the full sub-categories of $\mathrm{Per}(A)$ consisting of complete PERs and admissible PERs, respectively.

Definition 5. *Define $u : \mathbb{V} \rightarrow_c (T(\mathbb{V}) \rightarrow_c T(\mathbb{V}))$ as follows,*

$$u(x)(y) \overset{\textit{def}}{=} \begin{cases} z & \textit{if } x \cdot in_T(y) = in_T(z) \\ \bot & \textit{otherwise} \end{cases}$$

and let lfp denote the realizer $in_{\rightarrow}(\lambda x.\ [in_T(\sqcup_n (u(x))^n)])$.

Lemma 1. *Let $R \in \mathrm{AdmPer}(T(\mathbb{V}))$, then*

$$\textit{lfp} \in |(in_T(R) \rightarrow in_T(R)) \rightarrow in_T(R)|$$

and for all $\alpha \in |in_T(R) \rightarrow in_T(R)|$,

$$\alpha \cdot (\textit{lfp} \cdot \alpha)\ in_T(R)\ \textit{lfp} \cdot \alpha$$

Proof. See Lemma 6 in Appendix C.1, which generalizes the fixed point operator to impredicative Π types into an admissible PER.

The above development is standard and suffices to model fixed points over partial types in a non-stateful, simply-typed setting. However, it does not extend directly to a stateful dependently-typed setting: Assume $\vdash \tau : \boldsymbol{type}$ and $x : \tau \vdash \sigma : \boldsymbol{type}$. Then τ is interpreted as a PER $R \in \mathrm{Per}(\mathbb{V})$, and σ as an $[R]$-indexed family of PERs $S \in [R] \to \mathrm{Per}(\mathbb{V})$, and $\vdash \Sigma x : \tau.\sigma : \boldsymbol{type}$ as the PER $\Sigma_R(S)$:

$$\Sigma_R(S) = \{(in_\times(a_1, b_1), in_\times(a_2, b_2)) \mid a_1 \; R \; a_2 \wedge b_1 \; S([a_1]_R) \; b_2\}.$$

In general, this PER is not chain-complete even if R and each S_x is: given a chain $(a_i, b_i)_{i \in \mathrm{N}}$, we do not know in general that $[a_i]_R = [a_j]_R$ and hence cannot apply the completeness of S_x to the chain $(b_i)_{i \in \mathrm{N}}$ for any $x \in [R]$.

To rectify this problem we impose the following monotonicity condition on the PERs, which ensures exactly that $a_i \; R \; a_j$, for all $i, j \in \mathbb{N}$, and hence that $\Sigma_R(S)$ is chain-complete.

Definition 6 (CMPer(A))**.** *A PER $R \subseteq A \times A$ on a pre-domain A is* monotone *if, for all $x, y \in |R|$ such that $x \leq y$, we have $(x, y) \in R$. Let* CMPer(A) *denote the full sub-category of* Per(A) *on the complete monotone PERs.*

Restricting to complete monotone PERs forces a trivial equality on any particular Hoare type, as all of the elements of the type have to be equal to the diverging computation. However, it does not trivialize the totality of Hoare types, as we can still interpret each distinct Hoare type, $\boldsymbol{ST} \; \tau \; s$, as a distinct PER R, containing the computations that satisfy the specification s.

Restricting to complete monotone PERs does not collapse the equality on types in the purely functional fragment of iHTT, as these types are all interpreted as PERs without a bottom element in their domain. In particular, Π-types, which are modelled as elements of $\mathbb{V} \to \mathbb{V}_\perp$, picks out only those elements that map to non-bottom on their domain.

We shall see later that the monotonicity condition is also used to interpret partial types with a post-condition, which induces a dependency similar to that of Σ types, in the semantics of partial types.

3.1 iHTT

So far, we have informally introduced a PER model of a single dependent type universe extended with partial types. The next step is to scale the ideas to a model of all of iHTT, and to prove that we do indeed get a model of iHTT. We start by showing that CMPERs and assemblies form a model of the underlying dependent type theory. Next, we sketch the interpretation of iHTT specific features such as heaps and computations in the model. Lastly, we show that the model features W-types at both the ***set*** and ***type*** universe.

Underlying DTT. We begin by defining a general class of models for the dependent type theory underlying iHTT, and then present a concrete instance based on complete monotone PERs.

To simplify the presentation and exploit existing categorical descriptions of models of the Calculus of Constructions, the model will be presented using the fibred approach of [14]. To simplify the definition of the interpretation function, we consider a split presentation of the model (i.e., with canonical choice of all fibred structure, preserved on-the-nose).

Definition 7 (Split iHTT structure). *A split iHTT structure is a structure*

$$\begin{tikzcd}
\mathbb{C} \arrow[r, "\mathcal{I}_{prf}"] \arrow[dr, "r"'] & \mathbb{D} \arrow[r, "\mathcal{I}_{el}"] \arrow[d, "q"] & \mathbb{E} \arrow[r, "\mathcal{P}"] \arrow[d] & \mathbb{B}^{\to} \arrow[dl] \\
& \mathbb{B} \arrow[r, equal] & \mathbb{B} &
\end{tikzcd}$$

such that

– $\mathcal{P}$ *is a split closed comprehension category,*

– *$\mathcal{I}_{el}$ and $\mathcal{I}_{prf}$ are split fibred reflections*
– *the coproducts induced by the $\mathcal{I}_{el}$ reflection are strong (i.e., $\mathcal{P} \circ \mathcal{I}_{el}$ is a split closed comprehension category), and*
– *there exists objects $\Omega_{el}, \Omega_{prf} \in \mathbb{E}_1$ such that $\{\Omega_{el}\}$ is a split generic object for the q fibration and $\{\Omega_{prf}\}$ is a split generic object for the r fibration, where $\{-\} = \mathrm{Dom} \circ \mathcal{P} : \mathbb{E} \to \mathbb{B}$.*

The idea is to model contexts in $\mathbb{B}$, and the three universes, ***prop***, ***set***, and ***type*** in fibres of $\mathbb{C}$, $\mathbb{D}$ and $\mathbb{E}$, respectively. The split closed comprehension category structure models unit, Π and Σ types in the ***type*** universe. The split fibred reflections models the inclusion of ***prop*** into ***set*** and ***set*** into ***type*** and induces unit, Π and weak Σ types in ***prop*** and ***set***. Lastly, the split generic objects models the axioms ***prop*** : ***type*** and ***set*** : ***type***, respectively.

The concrete model we have in mind is mostly standard: the contexts and the ***type*** universe will be modelled with assemblies, the ***set*** universe with complete monotone PERs, and the ***prop*** universe with regular subobjects of assemblies. We begin by defining a category of uniform families of complete monotone PERs. Uniformity refers to the fact that each morphism is realized by a single $\alpha \in \mathbb{V}$.

Definition 8 **(**$\mathrm{UFam}(\mathrm{CMPer}(A))$**).**

– *Objects are pairs $(I, (S_i)_{i\in|I|})$ where $I \in \mathrm{Asm}(\mathbb{V})$ and each $S_i \in \mathrm{CMPer}(\mathbb{V})$.*
– *Morphisms from (I, S_i) to (J, T_j) are pairs $(u, (f_i)_{i\in|I|})$ where*

$$u : I \to J \in \mathrm{Asm}(\mathbb{V}) \qquad \textit{and} \qquad f_i : [S_i] \to [T_{u(i)}]$$

such that there exists an $\alpha \in \mathbb{V}$ satisfying

$$\forall i \in |I|.\ \forall e_i \in E_I(i).\ \forall e_v \in |S_i|.\ \alpha \cdot e_i \cdot e_v \in f_i([e_v]_{S_i})$$

Recall [14] that the standard category $\mathrm{UFam}(\mathrm{Per}(A))$ is defined in the same manner, using all PERs instead of only the complete monotone ones.

Let $\mathrm{RegSub}(\mathrm{Asm}(\mathbb{V}))$ denote the standard category of regular subobjects of assemblies and $\mathrm{UFam}(\mathrm{Asm}(\mathbb{V}))$ the standard category of uniform families of assemblies (see [14] for a definition). There is a standard split fibred reflection of $\mathrm{UFam}(\mathrm{Per}(A))$ into $\mathrm{UFam}(\mathrm{Asm}(A))$: the inclusion views a PER R as the assembly $([R], id_{[R]})$ [14]. This extends to a split fibred reflection of $\mathrm{UFam}(\mathrm{CMPer}(A))$ into $\mathrm{UFam}(\mathrm{Asm}(V))$ by composing with the following reflection of $\mathrm{UFam}(\mathrm{CMPer}(A))$ into $\mathrm{UFam}(\mathrm{Per}(A))$.

Lemma 2. *The inclusion $\mathcal{I} : \mathrm{UFam}(\mathrm{CMPer}(A)) \to \mathrm{UFam}(\mathrm{Per}(A))$ is a split fibred reflection.*

Proof (Sketch). We show that $\mathrm{CMPer}(A)$ is a reflective sub-category of $\mathrm{Per}(A)$; the same construction applies to uniform families, by a point-wise lifting. The left-ajoint, $\mathcal{R} : \mathrm{Per}(A) \to \mathrm{CMPer}(A)$ is given by monotone completion:

$$\mathcal{R}(S) = \overline{S} \qquad\qquad \mathcal{R}([\alpha]_{R\to S}) = [\alpha]_{\overline{R}\to\overline{S}}$$

where $\overline{R} \stackrel{\text{def}}{=} \bigcap\{S \in \mathrm{CMPer}(\mathbb{V}) \mid R \subseteq S\}$ for a PER $R \in \mathrm{Per}(\mathbb{V})$.

Let $S \in \mathrm{Per}(A)$ and $T \in \mathrm{CMPer}(A)$. Since the underlying realizers are continuous functions, we have that (see Lemma 7 in Appendix C)

$$\overline{S} \to T = S \to T$$

which induces the adjoint-isomorphism:

$$\mathrm{CMPer}(A)(\mathcal{R}(S), T) \cong [\overline{S} \to T] = [S \to T] \cong \mathrm{Per}(A)(S, \mathcal{I}(T))$$

Lemma 3. *The coproducts induced by the $\mathcal{I}_{el} : \mathrm{UFam}(\mathrm{CMPer}(\mathbb{V})) \to \mathrm{UFam}(\mathrm{Asm}(\mathbb{V}))$ reflection are strong.*

Proof. Let $I \in \mathrm{Asm}(\mathbb{V})$, $X \in \mathrm{UFam}(\mathrm{CMPer}(\mathbb{V}))_I$ and $Y \in \mathrm{UFam}(\mathrm{CMPer}(\mathbb{V}))_{\{X\}}$. Then the induced coproduct, $\Sigma_X(Y)$, is given by the family of complete monotone PERs,

$$\Sigma_X(Y) = \left(\overline{\left\{(\alpha,\beta) \mid \exists x, y \in \coprod_{x \in [X_i]} [Y_{(i,x)}].\ \alpha \in E_{\coprod_i}(x) \wedge \beta \in E_{\coprod_i}(y) \wedge x \sim_i y\right\}}\right)_{i \in |I|}$$

where

$$E_{\coprod_i}(a,b) = \{in_\times(d,e) \mid d \in a \wedge b \in e\}$$
$$x \smile_i y \quad \text{iff} \quad E_{\coprod_i}(x) \cap E_{\coprod_i}(y) = \emptyset$$

and $\sim_i$ is the transitive closure of $\smile_i$.

Since $\mathcal{I}_{el}(X)$ and $\mathcal{I}_{el}(Y)$ are modest sets, $\smile_i$ is the equality relation on $\coprod_{x\in[X_i]}[Y_{(i,x)}]$, and the above PER thus reduces to the monotone completion of the standard PER interpretation of Σ types.

$$\Sigma_X(Y) = (\overline{\{(in_\times(a_1,b_1), in_\times(a_2,b_2)) \mid a_1\ X_i\ a_2 \wedge b_1\ Y_{(i,[a_1]_{X_i})}\ b_2\}})_{i\in|I|}$$

Since each X_i and each $Y_{(i,x)}$ is a complete monotone PER, the standard PER interpretation of Σ types is already a complete monotone PER (see Lemma 8 in Appendix C) and thus,

$$\Sigma_X(Y) = (\{(in_\times(a_1,b_1), in_\times(a_2,b_2)) \mid a_1\ X_i\ a_2 \wedge b_1\ Y_{(i,[a_1]_{X_i})}\ b_2\})_{i\in|I|}$$

The coproducts thus coincide with the coproducts induced by the $\mathrm{UFam}(\mathrm{Per}(\mathbb{V})) \to \mathrm{UFam}(\mathrm{Asm}(\mathbb{V}))$ reflection, which are strong [14, Section 10.5.8].

Theorem 1. *The diagram below forms a split iHTT structure.*

$$\begin{tikzcd}
\mathrm{RegSub}(\mathrm{Asm}(\mathbb{V})) \arrow[r] \arrow[dr] & \mathrm{UFam}(\mathrm{CMPer}(\mathbb{V})) \arrow[l, bend right] \arrow[r] \arrow[d] & \mathrm{UFam}(\mathrm{Asm}(\mathbb{V})) \arrow[l, bend right] \arrow[r] \arrow[d] & \mathrm{Asm}(\mathbb{V})^{\to} \arrow[dl] \\
& \mathrm{Asm}(\mathbb{V}) \arrow[r, equal] & \mathrm{Asm}(\mathbb{V}) &
\end{tikzcd}$$

Interpretation. Except for impredicative Σ types and the axiom ***set*** : ***type***, the interpretation of the underlying dependent type theory in the above concrete split iHTT structure is exactly the standard PER-assembly interpretation. The type ***set*** is interpreted as the set of complete monotone PERs over $\mathbb{V}$, instead of the set of all PERs over $\mathbb{V}$. Impredicative Σ types are interpreted as the monotone completion of the standard PER-assembly interpretation. For completeness, we have written out the concrete interpretation of the underlying dependent type theory in Appendix B.

As terms generally have multiple typing derivations in dependent type theories, the interpretation function is typically defined as a partial function on pre-terms and pre-contexts, and later shown to be defined on well-typed terms and contexts [23]. Formally, the interpretation is given by three mutually recursive partial functions,

$$[\![-]\!]^{\mathrm{Ctx}} : \mathrm{Ctx} \rightharpoonup obj(\mathrm{Asm}(\mathbb{V}))$$
$$[\![-]\!]^{\mathrm{Type}} : \mathrm{Ctx} \times \mathrm{Term} \rightharpoonup obj(\mathrm{UFam}(\mathrm{Asm}(\mathbb{V})))$$
$$[\![-]\!]^{\mathrm{Term}} : \mathrm{Ctx} \times \mathrm{Term} \rightharpoonup hom(\mathrm{UFam}(\mathrm{Asm}(\mathbb{V})))$$

where Ctx is the set of pre-contexts and Term the set of pre-terms (defined in Appendix A.1), such that,

$$[\![\Gamma \vdash A]\!]^{\mathrm{Type}} \in \mathrm{UFam}(\mathrm{Asm}(\mathbb{V}))_{[\![\Gamma]\!]}^{\mathrm{Ctx}}$$
$$[\![\Gamma \vdash \mathrm{M}]\!]^{\mathrm{Term}} \in \mathrm{UFam}(\mathrm{Asm}(\mathbb{V}))_{[\![\Gamma]\!]}^{\mathrm{Ctx}}(1[\![\Gamma]\!]^{\mathrm{Ctx}}, [\![\Gamma \vdash A]\!]^{\mathrm{Type}})$$

for well-typed types, $\Gamma \vdash A : \boldsymbol{type}$, and well-typed terms, $\Gamma \vdash M : A$. Terms of small types, $\Gamma \vdash \mathrm{M} : \boldsymbol{el}(\tau)$, are thus interpreted as morphisms in UFam(CMPer($\mathbb{V}$)) fibres as follows:

$$\begin{aligned}
[\![\Gamma \vdash \mathrm{M} : \boldsymbol{el}(\tau)]\!]^{\mathrm{Term}} &\in \mathrm{UFam}(\mathrm{Asm}(\mathbb{V}))_I(1I, [\![\Gamma \vdash \boldsymbol{el}(\tau)]\!]^{\mathrm{Type}}) \\
&= \mathrm{UFam}(\mathrm{Asm}(\mathbb{V}))_I(1I, \mathcal{I}_{el}([\![\Gamma \vdash \tau]\!]^{\mathrm{Term}})) \\
&\cong \mathrm{UFam}(\mathrm{CMPer}(\mathbb{V}))_I(1I, [\![\Gamma \vdash \tau]\!]^{\mathrm{Term}})
\end{aligned}$$

where $I = [\![\Gamma]\!]^{\mathrm{Ctx}}$. Furthermore, global sections in UFam(CMPer($\mathbb{V}$)) fibres are uniquely determined by their realizers:

$$\mathrm{UFam}(\mathrm{CMPer}(\mathbb{V}))_I(1I, X) \cong \left[\bigcap_{i \in |I|} \Delta(E_I(i)) \to X_i\right]$$

for $I \in \mathrm{Asm}(\mathbb{V})$ and $X \in \mathrm{UFam}(\mathrm{CMPer}(\mathbb{V}))_I$. We can thus define the interpretation of terms of small types by giving the underlying realizer. This is the view we will employ when defining the interpretation of the basic computations of iHTT.

Heaps. Pre- and post-conditions in iHTT are expressed as predicates over a type ***heap***, of heaps. In addition to a constant denoting the empty heap, this type features the following operations: ***upd***, ***free***, ***max***, ***dom***, and ***peek***. ***upd*** and ***free*** updates and frees the value at a given location, respectively. ***peek*** returns the value (if any) at a given location. ***dom*** decides whether a given location is allocated and ***max*** returns the largest allocated location (and 0 for the empty heap). The ***max*** operation allows one to define a recursion operator on heaps. From these low-level operations we can define the high-level operations such as points-to ($\mapsto$) and disjoint union ($\bullet$) used in Section 2.

heap is a large type (i.e., ***heap*** : ***type***). The types of the built-in operations are as follows.

$$\begin{aligned}
\boldsymbol{empty} &: \boldsymbol{heap} \\
\boldsymbol{upd} &: \Pi\tau : \boldsymbol{set}.\ \boldsymbol{heap} \to \boldsymbol{el}(\boldsymbol{nat}) \to \boldsymbol{el}(\tau) \to \boldsymbol{heap} \\
\boldsymbol{free} &: \boldsymbol{heap} \to \boldsymbol{el}(\boldsymbol{nat}) \to \boldsymbol{heap} \\
\boldsymbol{max} &: \boldsymbol{heap} \to \boldsymbol{el}(\boldsymbol{nat}) \\
\boldsymbol{dom} &: \boldsymbol{heap} \to \boldsymbol{el}(\boldsymbol{nat}) \to \boldsymbol{bool} \\
\boldsymbol{peek} &: \boldsymbol{heap} \to \boldsymbol{el}(\boldsymbol{nat}) \to \boldsymbol{el}(1) + (\Sigma\tau : \boldsymbol{set}.\ \boldsymbol{el}(\tau))
\end{aligned}$$

We use $h[m \mapsto_\tau v]$ as shorthand for $\boldsymbol{upd}\ \tau\ h\ m\ v$ and $h \setminus m$ as shorthand for $\boldsymbol{free}\ h\ m$. These operations satisfy the heap axioms in Appendix A.11, which include several axioms relating the basic operations in addition to an induction principle and an extensionality principle for heaps.

We would like to model the values of ***heap*** as elements of $H(\mathbb{V})$. However, with this interpretation of ***heap*** we cannot interpret the update operation, $h[m \mapsto_\tau v]$, as the definition would not be independent of the choice of realizer for v. Rather, we introduce a notion of a world, which gives a notion of heap equivalence and interpret a heap as a pair consisting of a world and an equivalence class of heaps in the given world. A world is a finite map from locations to small semantic types:

$$\mathbb{W} \overset{\mathrm{def}}{=} \boldsymbol{ptr} \overset{\mathrm{fin}}{\rightharpoonup} \mathrm{CMPer}(\mathbb{V})$$

and two heaps $h_1, h_2 \in H(\mathbb{V})$ are considered equivalent in a world $w \in \mathbb{W}$ iff their support equals the domain of the world and their values are point-wise related by the world:

$$h_1 \sim_w h_2 \quad \text{iff} \quad supp(h_1) = supp(h_2) = \mathrm{Dom}(w) \wedge \forall l \in \mathrm{Dom}(w).\ h_1(l)\ w(l)\ h_2(l)$$

A typed heap is then a pair consisting of a world and an equivalence class of domain-theoretic heaps,

$$\mathbb{H}_t \stackrel{\text{def}}{=} \coprod_{w \in \mathbb{W}} [\sim_w]$$

and ***heap*** is interpreted as the set of typed heaps with the underlying domain-theoretic heaps as realizers:

$$[\![\Gamma \vdash \boldsymbol{heap}]\!]^{\text{Type}} = (\mathbb{H}_t, (w, U) \mapsto in_H(U))_{i \in |I|}$$

for $I = [\![\Gamma]\!]^{\text{Ctx}}$. That is, for each i, we have the assembly with underlying set $\mathbb{H}_t$ and with realizability map $\mathbb{H}_t \to P(\mathbb{V})$ given by $(w, U) \mapsto in_H(U)$. The realizers themselves do not have to contain any typing information, as we interpret small types with trivial realizability information (i.e., $[\![\Gamma \vdash \boldsymbol{set} : \boldsymbol{type}]\!] = \nabla(\text{CMPer}(\mathbb{V}))$). Let $I = [\![\Gamma]\!]$, then

$$\begin{aligned}
[\![\Gamma \vdash \boldsymbol{empty}]\!]_i^{\text{Term}} &= ([], \{[]\}) \\
[\![\Gamma \vdash \boldsymbol{upd}]\!]_i^{\text{Term}}(R)(w, [h]_w)([n])([v]) &= (w[n \mapsto R], [h[n \mapsto v]]_{w[n \mapsto R]}) \\
[\![\Gamma \vdash \boldsymbol{free}]\!]_i^{\text{Term}}(w, [h]_w)([n]) &= (w|_{\text{dom}(w) \setminus n}, [h|_{\text{dom}(w) \setminus n}]) \\
[\![\Gamma \vdash \boldsymbol{max}]\!]_i^{\text{Term}}(w, [h]_w) &= [in_{\mathbb{N}}(\max\{n \in \mathbb{N} \mid h(n) \neq \bot \vee n = 0\})] \\
[\![\Gamma \vdash \boldsymbol{dom}]\!]_i^{\text{Term}}(w, [h]_w)([n]) &= \begin{cases} [in_{\mathbb{N}}(0)] & \text{if } w(n) \text{ defined} \\ [in_{\mathbb{N}}(1)] & \text{otherwise} \end{cases} \\
[\![\Gamma \vdash \boldsymbol{peek}]\!]_i^{\text{Term}}(w, [h]_w)([n]) &= \begin{cases} inl(w(n), [h(n)]_{w(n)}) & \text{if } w(n) \text{ defined} \\ inr([*]) & \text{otherwise} \end{cases}
\end{aligned}$$

In the interpretation of update, the world is used to ensure that the interpretation is independent of the choice of realizer v for N_2. Since the underlying domain-theoretic heaps do not contain any typing information, the world is also used in the interpretation of peek, to determine the type of the value stored at the given location.

Note that iHTT has "strong" update and that the world is modified to contain the new type (semantically, the per R) upon update. Thus our notion and use of worlds is different from the use of worlds in models of "weak" ML-like reference types, e.g. [5]; in particular, note that we do not index every type by a world, but only use worlds to interpret the type of heaps and the operations thereon (see also the next subsection for further discussion).

Theorem 2. *The heap axioms in Appendix A.11 hold in the model.*

Hoare Types. We are now ready to sketch the interpretation of Hoare-types. The idea is to interpret Hoare-types as PERs on elements of $T(\mathbb{V})$ that satisfy the given specification, but with a trivial equality. Specifically, given a partial correctness specification, we define an admissible subset $X \subseteq T(\mathbb{V})$ of computations satisfying the specification, and interpret the associated Hoare type as the PER,

$$R = in_T(\Delta(X)) = \{(in_T(f), in_T(g)) \mid f \in X \wedge g \in X\}$$

The trivial equality ensures that R is trivially monotone and admissibility on the sub-domain of computations follows from admissibility of X.

Assume a semantic pre-condition $P \in \mathbb{H}_t \to 2$, a small semantic type $R \in \text{CMPer}(\mathbb{V})$, and a semantic post-condition $Q \in [R] \times \mathbb{H}_t \times \mathbb{H}_t \to 2$. As explained in the previous section, the pre- and post-condition is expressed in terms of a typed heaps, $\mathbb{H}_t$, instead of the underlying domain theoretic heaps. The subset of computations satisfying the specification is thus defined using the usual "forall initial worlds, there exists a terminal world"-formulation, known from models of ML-like references [5]. However, as iHTT supports strong update and deallocation, the terminal world is not required to be an extension of the initial world. Specifically, define $hoare(R, P, Q)$ as the following subset of $T(\mathbb{V})$:

$$\begin{aligned}hoare(R,P,Q) \overset{\text{def}}{=} \{f \in T(\mathbb{V}) \mid \forall w \in \mathbb{W}.\ \forall h \in |\sim_w|.\ P(w,[h]_w) = \top \Rightarrow \\ (f(h) = \bot\ \vee \exists v', h'.\ f(h) = (v', h') \wedge v' \in |R| \wedge \\ h' \in \overline{\{h' \in H(\mathbb{V}) \mid \exists w' \in \mathbb{W}.\ Q([v']_R)(w,[h]_w)(w',[h']_{w'}) = \top\}})\}\end{aligned}$$

where $\overline{(-)}$ denotes the chain-completion operator on $T(\mathbb{V})$. The explicit chain-completion of the post-condition is required because of the existential quantification over worlds. Furthermore, since the post-condition is indexed by the return value $[v']_R$, monotonicity is used to collapse a chain of domain-theoretic values $v_1 \leq v_2 \leq \cdots$ into a single type-theoretic value $[v_1]_R$, when proving that $hoare(R, P, Q)$ is an admissible subset of $T(\mathbb{V})$.

A Hoare-type in the is now interpreted as follows:

$$[\![\Gamma \vdash \textbf{\textit{st}}\ \tau\ (\mathrm{P},\mathrm{Q})]\!]_i^{\mathrm{Type}} = in_T(\Delta(hoare([\![\Gamma \vdash \tau]\!]_i, [\![\Gamma \vdash \mathrm{P}]\!]_i, [\![\Gamma \vdash \mathrm{Q}]\!]_i)))$$

The previous model of iHTT [21] featured a non-trivial computational equality. However, the previous model lacked the worlds introduced in the previous section and with it a useful notion of heap equivalence. As a result, the computational equality in the previous model was very strict, rendering the structural rule for existentials in Hoare logic unsound. The new model validates all the usual structural rules of Hoare-logic.

Computations. iHTT contains five basic computations for returning a value, reading from the heap, writing to the heap, allocating a location and deallocation a location. These basic computations are given by the following terms, where $\{\mathrm{P}\}\tau\{\mathrm{Q}\}$ is shorthand for $\textbf{\textit{st}}\ \tau\ (\mathrm{P},\mathrm{Q})^T$:

$$\begin{aligned}\textbf{\textit{ret}} &: \Pi\tau : \textbf{\textit{set}}.\ \Pi v : \tau.\ \{\lambda\_.\top\}\tau\{\lambda r, h_i, h_t.\ h_i = h_t \wedge r = v\} \\ \textbf{\textit{read}} &: \Pi\tau : \textbf{\textit{set}}.\ \Pi l : \textbf{\textit{nat}}.\ \{\lambda h.\ \textbf{\textit{dom}}\ h\ l = \textbf{\textit{true}}\}\tau\{\lambda r, h_i, h_t.\ h_i = h_t \wedge \textbf{\textit{peek}}\ h_t\ l = inr(\tau, r)\} \\ \textbf{\textit{write}} &: \Pi\tau : \textbf{\textit{set}}.\ \Pi l : \textbf{\textit{nat}}.\ \Pi v : \tau.\ \{\lambda h.\ \textbf{\textit{dom}}\ h\ n = \textbf{\textit{true}}\}\tau\{\lambda r, h_i, h_t.\ h_t = h_i[l \mapsto_\tau v]\} \\ \textbf{\textit{alloc}} &: \Pi\tau : \textbf{\textit{set}}.\ \Pi v : \tau.\ \{\lambda\_.\top\}\textbf{\textit{nat}}\{\lambda r, h_i, h_t.\ h_t = h_i[r \mapsto_\tau v] \wedge r \neq 0 \wedge \textbf{\textit{dom}}\ h_i\ r = \textbf{\textit{false}}\} \\ \textbf{\textit{dealloc}} &: \Pi l : \textbf{\textit{nat}}.\ \{\lambda h.\ \textbf{\textit{dom}}\ h\ l = \textbf{\textit{true}}\}1\{\lambda\_, h_i, h_t.\ h_t = h_i \setminus l\}\end{aligned}$$

In addition, iHTT contains a term, ***bind***, for combining a two computations in a sequential composition:

$$\begin{aligned}\textbf{\textit{bind}} : \Pi\tau,\sigma : \textbf{\textit{set}}.\ \Pi s_1 : spec\ \tau.\ \Pi s_2 : \tau \to spec\ \sigma.\ \textbf{\textit{st}}\ \tau\ s_1 \to (\Pi v : \tau.\ \textbf{\textit{st}}_\sigma\ (s_2\ v)) \to \\ \{\lambda h_i.\ \pi_1(s_1)\ h_i \wedge \forall x h_t.\ \pi_2(s_1)\ x\ h_i\ h_t \Rightarrow \pi_1(s_2\ x)\ h_m\} \\ \sigma \\ \{\lambda r, h_i, h_t.\ \exists x, h.\ \pi_2(s_1)\ x\ h_i\ h_t \wedge \pi_2(s_2)\ r\ h\ h_t\}\end{aligned}$$

and a term, ***do***, which corresponds to the structural rule of consequence in Hoare-logic:

$$\textbf{\textit{do}} : \Pi\tau : \textbf{\textit{set}}.\ \Pi s_1, s_2 : spec\ \tau.\ conseq\ s_1\ s_2 \to \textbf{\textit{st}}\ \tau\ s_1 \to \textbf{\textit{st}}\ \tau\ s_2$$

where *conseq* is specification implication:

$$\begin{aligned}&conseq : \Pi\tau : \textbf{\textit{set}}.\ spec\ \tau \to spec\ \tau \to \textbf{\textit{prop}} \\ &conseq = \lambda\tau.\ \lambda s_1.\ \lambda s_2.\ (\forall i.\ \pi_1(s_2)\ i \Rightarrow \pi_1(s_1)\ i) \wedge (\forall y, i, m.\ \pi_1(s_2)\ i \Rightarrow \pi_2(s_1)\ y\ i\ m \Rightarrow \pi_2(s_2)\ y\ i\ m)\end{aligned}$$

Lastly, for each type A iHTT contains a fixed point operator, $\textbf{\textit{fix}}_A$, for computations with an argument of type A:

$$\textbf{\textit{fix}}_A : \Pi\tau : \textbf{\textit{set}}.\ \Pi s : spec\ \tau.\ ((\Pi x : A.\ \textbf{\textit{st}}\ \tau\ s) \to (\Pi x : A.\ \textbf{\textit{st}}\ \tau\ s)) \to \Pi x : A.\ \textbf{\textit{st}}\ \tau\ s$$

Below we give the interpretation of the basic computations, by giving the underlying realizers. As a notational convenience, we write $\lambda x.\ f$ as shorthand for $in_{\rightarrow}(\lambda x.\ f)$. Let $I = [\![\Gamma]\!]$, then,

$$\begin{aligned}
[\![\Gamma \vdash \boldsymbol{ret}]\!] &= \lambda e_i.\ \lambda\_.\ \lambda e_v.\ in_T(\lambda h.\ (e_v, h)) \\
[\![\Gamma \vdash\ \boldsymbol{read}]\!] &= \lambda e_i.\ \lambda\_.\ \lambda n.\ in_T(\lambda h.\ \boldsymbol{if}\, h(n) \neq \bot\ \boldsymbol{then}\ (h(n), h)\ \boldsymbol{else\ err}) \\
[\![\Gamma \vdash \boldsymbol{write}]\!] &= \lambda e_i.\ \lambda\_.\ \lambda n.\ \lambda e_v.\ in_T(\lambda h.\ \boldsymbol{if}\, h(n) \neq \bot\ \boldsymbol{then}\ (in_1(*), h[n \mapsto e_v])\ \boldsymbol{else\ err}) \\
[\![\Gamma \vdash \boldsymbol{alloc}]\!] &= \lambda e_i.\ \lambda\_.\ \lambda e_v.\ in_T(\lambda h.\ \boldsymbol{let}\, l = \boldsymbol{leastfree}(h)\ \boldsymbol{in}\ (l, h[l \mapsto e_v])) \\
[\![\Gamma \vdash \boldsymbol{dealloc}]\!] &= \lambda e_i.\ \lambda n.\ \lambda h.\ in_T(\lambda h.\ \boldsymbol{if}\, h(n) = \bot\ \boldsymbol{then\ err\ else}\ (*, h[n \mapsto \bot])) \\
[\![\Gamma \vdash \boldsymbol{bind}]\!] &= \lambda e_i.\ \lambda\_.\ \lambda\_.\ \lambda\_.\ \lambda\_.\ \lambda m.\ \lambda n.\ in_T(bind(m, n)) \\
[\![\Gamma \vdash \boldsymbol{do}]\!] &= \lambda e_i.\ \lambda\_.\ \lambda\_.\ \lambda\_.\ \lambda\_.\ \lambda m.\ m \\
[\![\Gamma \vdash \boldsymbol{fix}_A]\!] &= \lambda e_i.\ \lambda\_.\ \lambda\_.\ \lambda m.\ in_{\rightarrow}(\lambda v.\ in_T((\sqcup_n (u(m))^n)(v)))
\end{aligned}$$

where

$$\begin{aligned}
bind(m, n) &\overset{\text{def}}{=} \lambda h.\ \boldsymbol{case}\ \pi_T(m)(h)\ \boldsymbol{of} \\
&\qquad (v, h') \Rightarrow \pi_T(n \cdot v)(h') \\
&\qquad \boldsymbol{err} \quad\ \Rightarrow \boldsymbol{err} \\
&\qquad \bot \qquad \Rightarrow \bot \\
\boldsymbol{leastfree} &\overset{\text{def}}{=} \lambda h.\ \min\{n \in \mathbb{N}_+ \mid h(n) = \bot\} \\
u(x : \mathbb{V})(y : \mathbb{V} \to T(\mathbb{V}))(z : \mathbb{V}) &\overset{\text{def}}{=} \begin{cases} a & \text{if } x \cdot in_{\rightarrow}(\lambda z.\ in_T(y(z))) \cdot z = in_T(a) \\ \bot & \text{otherwise} \end{cases} \\
\pi_T(x : \mathbb{V}) &\overset{\text{def}}{=} \begin{cases} a & \text{if } x \downarrow \text{ and } x = in_T(a) \\ \bot_{T(\mathbb{V})} & \text{otherwise} \end{cases}
\end{aligned}$$

Theorem 3. *The interpretation of computations is sound, i.e., well-typed computations satisfy their specifications.*

Proof. We prove three representative cases: ***bind***, ***write***, and $\boldsymbol{fix}_A$.

In the case of ***bind***, the intermediate state in the execution of a sequential composition satisfies the chain-completion of the post-condition of the first computation. To show that ***bind*** is sound, one uses that the underlying computations are continuous to reason about intermediate states added via the chain-completion. Soundness follows from Lemma 9 in Appendix C.3.

The soundness of $\boldsymbol{fix}_A$ follows from Lemma 6 in Appendix C.1, which generalizes the the least fixed point operator on Per($\mathbb{V}$) to computations with a single argument of a large type.

Lastly, the soundness of ***write*** follows from Lemma 10 in Appendix C.3.

W-types. The presentation of (co)-inductive types in CIC is based on an intricate syntactic scheme of inductive families. So far, this presentation of (co)-inductive types has eluded a categorical semantics. Martin-Löf type theory features an alternative presentation, based on W-types (a type-theoretic formalization of well-founded trees), which is strong enough to represent a wide range of inductive types in extensional models (such as ours) [11, 3]. Since W-types in addition have a simple categorical semantics, we have chosen to show that iHTT models inductive types by showing that it models W-types. Specifically, we show that it models W-types at both the ***type*** and ***set*** universe, and that the W-types at the ***set*** universe supports elimination over large types.

Semantically, in the setting of locally cartesian closed categories, W-types are modelled as initial algebras of polynomial functors [16]. In the setting of split closed comprehension categories we define:

Definition 9. *A split closed comprehension category* $\mathcal{P} : \mathbb{E} \to \mathbb{B}^{\rightarrow}$ *has split W-types, if for every* $I \in \mathbb{B}$, $X \in \mathbb{E}_I$ *and* $Y \in \mathbb{E}_{\{X\}}$ *the endo-functor,*

$$P_{I,X,Y} = \Sigma_X \circ \Pi_Y \circ (\pi_X \circ \pi_Y)^* : \mathbb{E}_I \to \mathbb{E}_I$$

has a chosen initial algebra $\alpha_{I,X,Y} : P_{I,X,Y}(W_{I,X,Y}) \to W_{I,X,Y} \in \mathbb{E}_I$*, which is preserved on-the-nose by re-indexing functors.*

As is well-known, $\mathrm{Asm}(\mathbb{V})$ is locally cartesian closed and all polynomial functors on $\mathrm{Asm}(\mathbb{V})$ have initial algebras. This yields initial algebras for functors $P_{X,Y} : \mathrm{UFam}(\mathrm{Asm}(\mathbb{V}))_1 \to \mathrm{UFam}(\mathrm{Asm}(\mathbb{V}))_1$ for $X \in \mathbb{E}_1$ and $Y \in \mathbb{E}_{\{X\}}$. This lifts to an arbitrary context $I \in \mathrm{Asm}(\mathbb{V})$ by a point-wise construction, which yields split W-types, as re-indexing in $\mathrm{UFam}(\mathrm{Asm}(\mathbb{V}))$ is by composition:

Lemma 4. *The split ccomp* $\mathcal{P} : \mathrm{UFam}(\mathrm{Asm}(\mathbb{V})) \to \mathrm{Asm}(\mathbb{V})^{\to}$ *has split W-types.*

Proof. W-types in $\mathrm{Asm}(\mathbb{V})$ are constructed from the W-types in Sets, by restricting to "heriditarily realized" trees (see Appendix C.4 for the full construction): In the case where $I = 1$, assume $X \in \mathrm{Asm}(\mathbb{V})$ and $Y \in \mathrm{UFam}(\mathrm{Asm}(\mathbb{V}))_X$ and let $T_{X,Y}$ denote a solution to the set-isomorphism,

$$sup_{X,Y} : \coprod_{x \in |X|} |Y_x| \to T_{X,Y} \cong T_{X,Y}$$

and let $E_{X,Y} : T_{X,Y} \to \mathcal{P}(\mathbb{V})$ denote the unique function satisfying,

$$E_{X,Y}(sup(x,f)) = \{in_\times(e_x, e_f) \mid e_x \in E_X(x) \wedge \forall y \in |Y_x|.\ \forall e_y \in E_{Y_x}(y).\ e_f \cdot e_y \in E_{X,Y}(f(y))\}$$

Then $W_{X,Y} \overset{\text{def}}{=} (\{w \in T_{X,Y} \mid E_{X,Y}(w) \neq \emptyset\}, E_{X,Y}) \in \mathrm{Asm}(\mathbb{V})$ and $sup : P_{X,Y}(W_{X,Y}) \to W_{X,Y}$ is an initial $P_{X,Y}$ algebra, realized by the identity (Lemma 14 in Appendix C.4).

In the case of an arbitrary context $I \in \mathrm{Asm}(\mathbb{V})$, assume $X \in \mathrm{UFam}(\mathrm{Asm}(\mathbb{V}))_I$ and $Y \in \mathrm{UFam}(\mathrm{Asm}(\mathbb{V}))_{\{X\}}$, then we define $W_{I,X,Y}$ and $\alpha_{I,X,Y}$ by a point-wise lifting:

$$W_{I,X,Y} = \left(W_{X_i, \lambda y.\ Y_{(i,y)}}\right)_{i \in |I|}$$

and

$$\alpha_{I,X,Y} = \left(id_I, \left(sup_{X_i, \lambda y.\ Y_{(i,y)}}\right)_{i \in |I|}\right)$$

$\alpha_{I,X,Y}$ is uniformly realized, since each of the underlying *sup* functions is realized by the identitiy.

To show that these are further preserved on the nose by reindexing, assume $J \in \mathrm{Asm}(\mathbb{V})$ and $u : J \to I \in \mathrm{Asm}(\mathbb{V})$. Then,

$$u^*(\alpha_{I,X,Y}) = \left(id_J, \left(\alpha_{X_{u(j)}, \lambda y.\ Y_{(u(j),y)}}\right)_{j \in |J|}\right) = \alpha_{J, u^*(X), \{\overline{u}(X)\}^*(Y)}$$

This models W-types in the ***type***-universe. In addition, iHTT features small W-types over small types, with elimination over large types. The idea is to model these small W-types by forming the W-type in $\mathrm{UFam}(\mathrm{Asm}(\mathbb{V}))$ and mapping it back into $\mathrm{UFam}(\mathrm{CMPer}(\mathbb{V}))$ using the reflection. The reflection works by collapsing an assembly into a PER by equating values with overlapping sets of realizers and into a complete monotone PER by taking the monotone completion of said PER. In the case where X and Y are in the image of $\mathcal{I}_{el}$, the construction of $W_{I,X,Y}$ sketched above yields a modest set (i.e., has no overlapping sets of realizers) and furthermore, the induced PER is already monotone and complete. This induces the following isomorphism, which allows us to model small W-types with elimination over large types:

Lemma 5. *For every* $I \in \mathbb{B}$*,* $X \in \mathrm{UFam}(\mathrm{CMPer}(\mathbb{V}))_I$*, and* $Y \in \mathrm{UFam}(\mathrm{CMPer}(\mathbb{V}))_{\{X\}}$*, there is a chosen isomorphism,* $W_{I,\mathcal{I}_{el}(X),\mathcal{I}_{el}(Y)} \cong \mathcal{I}_{el}(\mathcal{R}_{el}(W_{I,\mathcal{I}_{el}(X),\mathcal{I}_{el}(Y)}))$*, which is preserved on-the-nose by reindexing functors.*

Proof (Sketch). There is a well-known equivalence between PERs and modest sets. This extends to complete monotone PERs as follows: Define CMod($\mathbb{V}$) as the full-subcategory of Mod($\mathbb{V}$), consiting of modest sets (X, E_X), satisfying the following two conditions,

$$\forall x, y \in X.\ (\exists a, b.\ a \in E_X(x) \wedge b \in E_X(y) \wedge a \leq b) \Rightarrow x = y$$
$$\forall x \in X.\ \forall c : \mathbb{N} \rightarrow_m \mathbb{V}.\ (\forall n \in \mathbb{N}.\ c(n) \in E_X(x)) \Rightarrow \sqcup_n c(n) \in E_X(x)$$

then $\mathcal{I}_{el}$ and $\mathcal{R}_{el}$ forms an equivalence between CMPer($\mathbb{V}$) and CMod($\mathbb{V}$) and furthermore for every $X \in$ CMod($\mathbb{V}$) there is a chosen isomorphism $X \cong \mathcal{I}_{el}(\mathcal{R}_{el}(X))$. This lifts to a fibred equivalence between UFam(CMPer($\mathbb{V}$)) and UFam(CMod($\mathbb{V}$)), with chosen isomorphisms, $X \cong \mathcal{I}_{el}(\mathcal{R}_{el}(X))$ for $X \in \text{UFam}(\text{CMod}(\mathbb{V}))_I$, preserved on the nose by re-indexing functors.

The result thus follows by showing that $W_{X,Y} \in \text{CMPer}(\mathbb{V})$ for $X \in \text{Asm}(\mathbb{V})$ and $Y \in \text{UFam}(\text{Asm}(\mathbb{V}))_X$, which follows by well-founded induction on $T_{X,Y}$ (Lemma 15 in Appendix C.4).

4 Implementation

One of the advantages of weakening Hoare types instead of the underlying dependent type theory – as in the case of [21] – is that it can simplify implementation of the resulting type theory. In our case, presenting iHTT as an extension of CIC allows for easy implementation as an axiomatic extension of the Coq proof assistant.

Our implementation is based on the Coq infrastructure developed by Nanevski et. al. [19], to support efficient reasoning about stateful computations in Coq. This infrastructure defines a new Hoare type on top of the built-in Hoare type with support for more efficient reasoning, based on ideas from separation logic. Specifically, as illustrated in Section 2, this new Hoare type features (1) small footprint specifications and (2) efficient reasoning about heaps. Efficient reasoning about heaps is achieved by reasoning using the partial commutative monoid $(1 + \textbf{\textit{heap}}, \bullet)$ instead of the total non-commutative monoid $(\textbf{\textit{heap}}, \bullet)$, where ***heap*** is the actual type of heaps and $\bullet$ is heap union [19].

Compared to Nanevski et. al.'s implementation of predicative Hoare Type Theory (pHTT), this new implementation features higher-order store and impredicative quantification, but the predicative hierarchy lacks Hoare-types. The new implementation is almost source compatible with verifications in pHTT that do not exploit the predicative hierarchy.

Compared to the Ynot implementation of pHTT [18], in addition to impredicativity the main difference lies in the treatment of ghost variables. Post conditions in Ynot are unary and thus employ ghost-variables to relate the pre- and post-condition. Ynot expresses ghost variables of a specification as computationally irrelevant arguments to the computation. As Coq lacks support for computationally irrelevant variables, Ynot extends Coq with an injectivity axiom, which gives an embedding of ***set*** into the computationally irrelevant ***prop***-universe. This axiom is inconsistent with a proof irrelevant ***prop***-universe and thus in particular unsound in our model. Additionally, this limits Ynot's ghost variables to small types, whereas iHTT supports ghost variables of large types.

5 Related work

Our approach to partiality is based on the idea of partial types, as introduced by Constable and Smith [9]. We have already discussed its relation to the work on admissibility by Crary [10] in the introduction. Below we first discuss related work on partiality, followed by related work on partial correctness reasoning.

Bove and Capretta [7] proposed representing a partial function $f : A \rightharpoonup B$ as a total function $\overline{f} : \Pi a : A.\ P(a) \rightarrow B$, defined by recursion over an inductively defined predicative $P : A \rightarrow \textbf{\textit{prop}}$, expressing the domain of the partial function. This allows the definition of partial computations by general recursion, but does not model non-termination, as $\overline{f}$ can only be applied to arguments on which it terminates. Capretta [8] proposed an alternative co-inductive representation, which does model non-termination, representing a partial function $f : A \rightharpoonup B$ as a total function $\overline{f} : A \to B^v$, where B^v is co-inductive type of partial elements of type B. This representation yields a least fixed point operator on finitrary (continuous) endo-functions on $A \to B^v$. Capretta does not provide an fixed point induction principle, but we believe such a principle would require admissibility proofs.

Another alternative approach to partiality is to give a model of a language featuring general recursion inside a dependent type theory. This allows one to model and reason about partial computations inside the type theory, but does not extend the type theory itself with partial computations. This approach has for instance been studied and implemented by Reus et. al. [22], who formalized Synthetic Domain Theory in the Lego proof checker. The resulting type theory can be seen as a very expressive version of LCF. The synthetic approach alleviates the need for continuity proofs, but still requires admissibility proofs when reasoning by fixed point induction.

Hoare-style specification logics is another line of closely related work. With a collapsed computational equality, reasoning about a partial computation in iHTT is limited to Hoare-style partial correctness reasoning, as in a specification logic. With the modularity features provided by the underlying dependent type theory (i.e., Σ types), iHTT can thus be seen as a modular specification logic for a higher-order programming language.

6 Conclusion

We have presented a new approach for extending dependent type theory with potentially non-terminating computations, without weakening the underlying dependent type theory or adding restrictions on the use of fixed points in defining partial computations. We have shown that it scales to very expressive dependent type theories and effects beyond partiality, by extending the Calculus of Inductive Constructions with stateful and potentially non-terminating computations. We have further demonstrated that this approach is practical, by implementing this extension of CIC as an axiomatic extension of the Coq proof assistant.

To justify the soundness of our extension of CIC, we have presented a realizability model of the theory. For lack of space, we have limited the presentation to a single predicative universe, but the model can be extended to the whole predicative hierarchy $\boldsymbol{type} \subseteq \boldsymbol{type}_1 \subseteq \ldots$ of CIC.

References

1. The Coq Proof Assistant. http://coq.inria.fr/.
2. *Coq Reference Manual, Version 8.3.*
3. M. Abbott, T. Altenkirch, and N. Ghani. Representing nested inductive types using W-types. In *Proc. of ICALP*, 2004.
4. R. M. Amadio. Recursion over Realizability Structures. *Information and Computation*, 91:55–85, 1991.
5. L. Birkedal, K. Støvring, and J. Thamsborg. Realisability semantics of parametric polymorphism, general references and recursive types. *Math. Struct. Comp. Sci.*, 20(4):655–703, 2010.
6. D. Bjørner and C. B. Jones, editors. *The Vienna Development Method: The Meta-Language*, volume 61 of *Lecture Notes in Computer Science*. Springer, 1978.
7. A. Bove. Simple General Recursion in Type Theory. *Nordic Journal of Computing*, 8, 2000.
8. V. Capretta. General Recursion via Coinductive Types. *Logical Methods in Computer Science*, 1(2):1–28, 2005.
9. R. L. Constable and S. F. Smith. Partial Objects in Constructive Type Theory. In *Proceedings of Second IEEE Symposium on Logic in Computer Science*, 1987.
10. K. Crary. Admissibility of Fixpoint Induction over Partial Types. In *Automated Deduction - CADE-15*, 1998.
11. P. Dybjer. Representing inductively defined sets by wellorderings in Martin-Löf's type theory. *Theor. Comput. Sci.*, 176(1-2):329–335, 1997.
12. G. Gonthier and A. Mahboubi. A Small Scale Reflection Extension for the Coq system. Technical report, INRIA, 2007.
13. C. A. R. Hoare. Proof of correctness of data representations. *Acta Informatica*, 1:271–281, 1972.
14. B. Jacobs. *Categorical Logic and Type Theory.* Elsevier Science, 1999.
15. B. Meyer. *Object-oriented software construction.* Prentice Hall, 1997.

16. I. Moerdijk and E. Palmgren. Wellfounded trees in categories. *Annals of Pure and Applied Logic*, 104(1–3):189–218, 2000.
17. A. Nanevski, G. Morrisett, and L. Birkedal. Hoare Type Theory, Polymorphism and Separation. *Journal of Functional Programming*, 18(5–6):865–911, 2008.
18. A. Nanevski, G. Morrisett, A. Shinnar, P. Govereau, and L. Birkedal. Ynot: Dependent Types for Imperative Programs. In *Proceedings of ICFP 2008*, pages 229–240, 2008.
19. A. Nanevski, V. Vafeiadis, and J. Berdine. Structuring the Verification of Heap-Manipulating Programs. In *Proceedings of POPL 2010*, 2010.
20. P. O'Hearn, J. Reynolds, and H. Yang. Local reasoning about programs that alter data structures. In *CSL'01*, pages 1–19, 2001.
21. R. Petersen, L. Birkedal, A. Nanevski, and G. Morrisett. A Realizability Model of Impredicative Hoare Type Theory. In *Proceedings of ESOP 2008*, 2008.
22. B. Reus. Synthetic Domain Theory in Type Theory: Another Logic of Computable Functions. In *Proceedings of TPHOL 1996*, 1996.
23. T. Streicher. *Semantics of Type Theory: Correctness, Completeness, and Independence Results.* Birkhaeuser Verlag, 1991.

A Impredicative Hoare Type Theory

A.1 Syntax

$$
\begin{array}{rcl}
\mathit{\Gamma} & ::= & \varepsilon \mid \mathit{\Gamma}, x : A \\
\mathrm{M}, \mathrm{N}, A, B, \tau, \sigma, \mathrm{P}, \mathrm{Q} & ::= & x \mid \boldsymbol{set} \mid \boldsymbol{prop} \mid \boldsymbol{el}(\tau) \mid \boldsymbol{prf}(\mathrm{P}) \\
& \mid & \mathit{\Pi}^T x : A.B \mid \mathit{Abs}^T_{x:A.B}(\mathrm{M}) \mid \mathit{App}^T_{x:A.B}(\mathrm{M}, \mathrm{N}) \\
& \mid & \mathit{\Pi}^S x : A.\tau \mid \mathit{Abs}^S_{x:A.\tau}(\mathrm{M}) \mid \mathit{App}^S_{x:A.\tau}(\mathrm{M}, \mathrm{N}) \\
& \mid & \forall x : A.\mathrm{P} \mid \mathit{Abs}^P_{x:A.\mathrm{P}}(\mathrm{M}) \mid \mathit{App}^P_{x:A.\mathrm{P}}(\mathrm{M}, \mathrm{N}) \mid \bullet_{\mathrm{P}} \\
& \mid & \mathit{\Sigma}^T x : A.B \mid \mathit{Pair}^T_{x:A.B}(\mathrm{M}, \mathrm{N}) \mid \pi^1_{x:A.B}(\mathrm{M}) \mid \pi^2_{x:A.B}(\mathrm{M}) \\
& \mid & \mathit{\Sigma}^S x : A.\tau \mid \mathit{Pair}^S_{x:A.\tau}(\mathrm{M}, \mathrm{N}) \mid \boldsymbol{unpack}\ \mathrm{M}\ \boldsymbol{as}\ (x, y)\ \boldsymbol{in}\ \mathrm{N} \\
& \mid & A + B \mid \boldsymbol{inl}_{A,B}(\mathrm{M}) \mid \boldsymbol{inr}_{A,B}(\mathrm{M}) \mid \boldsymbol{case}_{A,B}\ \mathrm{M}\ \boldsymbol{in}\ \boldsymbol{inl}(x) \Rightarrow \mathrm{N}_1 \mid \boldsymbol{inr}(x) \Rightarrow \mathrm{N}_2 \\
& \mid & \mathcal{W}^T x : A.B \mid \mathit{sup}^T_{x:A.B}(\mathrm{M}) \mid \mathit{fold}^T_{x:A.B}(\mathrm{M}) \\
& \mid & \mathcal{W}^S x : \tau.\sigma \mid \mathit{sup}^S_{x:\tau.\sigma}(\mathrm{M}) \mid \mathit{fold}^S_{x:\tau.\sigma}(\mathrm{M}) \\
& \mid & 1 \mid () \\
& \mid & \boldsymbol{bool} \mid \boldsymbol{true} \mid \boldsymbol{false} \mid \boldsymbol{if}\ \mathrm{M}\ \boldsymbol{then}\ \mathrm{N}_1\ \boldsymbol{else}\ \mathrm{N}_2 \\
& \mid & \boldsymbol{nat} \mid \boldsymbol{zero} \mid \boldsymbol{succ}\ \mathrm{M} \mid \mathrm{M} == \mathrm{N} \mid \mathrm{M} < \mathrm{N} \\
& \mid & \boldsymbol{heap} \mid \boldsymbol{empty} \mid \boldsymbol{upd} \mid \boldsymbol{free} \mid \boldsymbol{max} \mid \boldsymbol{dom} \mid \boldsymbol{peek} \\
& \mid & \boldsymbol{st} \mid \boldsymbol{ret} \mid \boldsymbol{read} \mid \boldsymbol{write} \mid \boldsymbol{alloc} \mid \boldsymbol{dealloc} \mid \boldsymbol{bind} \mid \boldsymbol{do} \mid \boldsymbol{fix}_A
\end{array}
$$

A.2 Judgments

$\mathit{\Gamma}$ Ctx	$\mathit{\Gamma}$ is a well-formed context
$\mathit{\Gamma} \vdash A : \boldsymbol{type}$	A is a type in context $\mathit{\Gamma}$
$\mathit{\Gamma} \vdash \mathrm{M} : A$	M is a term of type A in context $\mathit{\Gamma}$
$\mathit{\Gamma} \vdash A = B : \boldsymbol{type}$	A and B are convertible in context $\mathit{\Gamma}$
$\mathit{\Gamma} \vdash \mathrm{M} = \mathrm{N} : A$	M and N are convertible in context $\mathit{\Gamma}$

We reserve the meta-variables A and B for types, the meta-variables τ and σ for small types (terms of type ***set***), and the meta-variables P and Q for propositions (terms of type ***prop***).

A.3 Contexts

$$\frac{}{\varepsilon \text{ Ctx}} \qquad\qquad \frac{x \notin \mathit{\Gamma} \qquad \mathit{\Gamma} \vdash A : \boldsymbol{type}}{\mathit{\Gamma}, x : A \text{ Ctx}}$$

A.4 Structural Rules

$$\frac{\mathit{\Gamma} \vdash A : \boldsymbol{type}}{\mathit{\Gamma}, x : A \vdash x : A}\,\pi \qquad \frac{\mathit{\Gamma} \vdash \mathrm{M} : A \qquad \mathit{\Gamma}, x : A, \mathit{\Delta} \vdash \mathcal{J}}{\mathit{\Gamma}, \mathit{\Delta}[\mathrm{M}/x] \vdash \mathcal{J}[\mathrm{M}/x]}\,\textsc{sub} \qquad \frac{\mathit{\Gamma} \vdash A : \boldsymbol{type} \qquad \mathit{\Gamma} \vdash \mathcal{J}}{\mathit{\Gamma}, x : A \vdash \mathcal{J}}\,\textsc{weak}$$

$$\frac{\mathit{\Gamma}, x : A, y : A, \mathit{\Delta} \vdash \mathcal{J}}{\mathit{\Gamma}, x : A, \mathit{\Delta}[x/y] \vdash \mathcal{J}[x/y]}\,\textsc{cont} \qquad \frac{\mathit{\Gamma} \vdash B : \boldsymbol{type} \qquad \mathit{\Gamma}, x : A, y : B, \mathit{\Delta} \vdash \mathcal{J}}{\mathit{\Gamma}, y : B, x : A, \mathit{\Delta} \vdash \mathcal{J}}\,\textsc{ex}$$

$$\frac{\mathit{\Gamma} \vdash \mathrm{M} : A \qquad \mathit{\Gamma} \vdash A = B : \boldsymbol{type}}{\mathit{\Gamma} \vdash \mathrm{M} : B}\,\textsc{conv}$$

A.5 Types

$$\frac{}{\varepsilon \vdash \textbf{\textit{set}} : \textbf{\textit{type}}}\ \text{TYPE} \qquad \frac{}{\varepsilon \vdash \textbf{\textit{prop}} : \textbf{\textit{type}}}\ \text{PROP}$$

$$\frac{\Gamma \vdash \tau : \textbf{\textit{set}}}{\Gamma \vdash \textbf{\textit{el}}(\tau) : \textbf{\textit{type}}}\ \text{ElT} \qquad \frac{\Gamma, x : A \vdash B : \textbf{\textit{type}}}{\Gamma \vdash \Pi^T x : A.\ B : \textbf{\textit{type}}}\ \Pi T \qquad \frac{\Gamma, x : A \vdash B : \textbf{\textit{type}}}{\Gamma \vdash \Sigma^T x : A.\ B : \textbf{\textit{type}}}\ \Sigma T$$

A.6 Small types

$$\frac{}{\Gamma \vdash 1 : \textbf{\textit{set}}}\ 1 \qquad \frac{}{\Gamma \vdash \textbf{\textit{bool}} : \textbf{\textit{set}}}\ \text{BOOL} \qquad \frac{}{\Gamma \vdash \textbf{\textit{nat}} : \textbf{\textit{set}}}\ \text{NAT} \qquad \frac{\Gamma \vdash \mathrm{P} : \textbf{\textit{prop}}}{\Gamma \vdash \textbf{\textit{prf}}(\mathrm{P}) : \textbf{\textit{set}}}\ \text{PrfT}$$

$$\frac{\Gamma, x : A \vdash \tau : \textbf{\textit{set}}}{\Gamma \vdash \Pi^S x : A.\ \tau : \textbf{\textit{set}}}\ \Pi S \qquad \frac{\Gamma, x : A \vdash \tau : \textbf{\textit{set}}}{\Gamma \vdash \Sigma^S x : A.\ \tau : \textbf{\textit{set}}}\ \Sigma S$$

A.7 Propositions

$$\frac{\Gamma, x : A \vdash \mathrm{P} : \textbf{\textit{prop}}}{\Gamma \vdash \forall x : A.\mathrm{P} : \textbf{\textit{prop}}}$$

$$\frac{\Gamma, x : A \vdash \mathrm{M} : \textbf{\textit{el}}(\textbf{\textit{prf}}(\mathrm{P}))}{\Gamma \vdash \lambda^P x : A.\mathrm{M} : \textbf{\textit{el}}(\textbf{\textit{prf}}(\forall x : A.\mathrm{P}))} \qquad \frac{\Gamma \vdash \mathrm{M} : \textbf{\textit{el}}(\textbf{\textit{prf}}(\forall x : A.\mathrm{P})) \qquad \Gamma \vdash \mathrm{N} : A}{\Gamma \vdash App^P_{x:A.\mathrm{P}}(\mathrm{M}, \mathrm{N}) : \textbf{\textit{el}}(\textbf{\textit{prf}}(\mathrm{P}[\mathrm{N}/x]))}$$

Using universal quantification, we define the usual logical connectives as follows:

$$\begin{aligned}
\mathrm{P} \Rightarrow \mathrm{Q} &\overset{\text{def}}{=} \forall x : \textbf{\textit{el}}(\textbf{\textit{prf}}(\mathrm{P})).\ \mathrm{Q} \\
\top &\overset{\text{def}}{=} \forall \mathrm{P} : \textbf{\textit{prop}}.\ \mathrm{P} \Rightarrow \mathrm{Q} \\
\bot &\overset{\text{def}}{=} \forall \mathrm{P} : \textbf{\textit{prop}}.\ \mathrm{P} \\
\mathrm{P} \wedge \mathrm{Q} &\overset{\text{def}}{=} \forall \mathrm{R} : \textbf{\textit{prop}}.\ (\mathrm{P} \Rightarrow \mathrm{Q} \Rightarrow \mathrm{R}) \Rightarrow \mathrm{R} \\
\mathrm{P} \vee \mathrm{Q} &\overset{\text{def}}{=} \forall \mathrm{R} : \textbf{\textit{prop}}.\ (\mathrm{P} \Rightarrow \mathrm{R}) \Rightarrow (\mathrm{Q} \Rightarrow \mathrm{R}) \Rightarrow \mathrm{R} \\
\neg \mathrm{P} &\overset{\text{def}}{=} \mathrm{P} \Rightarrow \bot \\
\exists x : A.\mathrm{P} &\overset{\text{def}}{=} \forall \mathrm{R} : \textbf{\textit{prop}}.\ (\Pi^P x : A.\ (\mathrm{P} \Rightarrow \mathrm{R})) \Rightarrow \mathrm{R} \\
a =_A b &\overset{\text{def}}{=} \forall \mathrm{P} : A \to \textbf{\textit{prop}}.\ \mathrm{P}(a) \Rightarrow \mathrm{P}(b)
\end{aligned}$$

A.8 Terms

$$\frac{}{\Gamma \vdash () : \textbf{\textit{el}}(1)}$$

$$\frac{}{\Gamma \vdash \textbf{\textit{true}} : \textbf{\textit{el}}(\textbf{\textit{bool}})} \qquad \frac{}{\Gamma \vdash \textbf{\textit{false}} : \textbf{\textit{el}}(\textbf{\textit{bool}})} \qquad \frac{\Gamma \vdash \mathrm{M} : \textbf{\textit{bool}} \qquad \Gamma \vdash \mathrm{N}_1 : A \qquad \Gamma \vdash \mathrm{N}_2 : A}{\Gamma \vdash \textbf{\textit{if}}\ \mathrm{M}\ \textbf{\textit{then}}\ \mathrm{N}_1\ \textbf{\textit{else}}\ \mathrm{N}_2 : A}$$

$$\frac{}{\Gamma \vdash \boldsymbol{zero} : \boldsymbol{el}(\boldsymbol{nat})} \qquad \frac{\Gamma \vdash \mathrm{M} : \boldsymbol{el}(\boldsymbol{nat})}{\Gamma \vdash \boldsymbol{succ}\ \mathrm{M} : \boldsymbol{el}(\boldsymbol{nat})}$$

$$\frac{\Gamma \vdash \mathrm{M}, \mathrm{N} : \boldsymbol{el}(\boldsymbol{nat})}{\Gamma \vdash \mathrm{M} == \mathrm{N} : \boldsymbol{el}(\boldsymbol{bool})} \qquad \frac{\Gamma \vdash \mathrm{M}, \mathrm{N} : \boldsymbol{el}(\boldsymbol{nat})}{\Gamma \vdash \mathrm{M} < \mathrm{N} : \boldsymbol{el}(\boldsymbol{bool})}$$

$$\frac{\Gamma, x : A \vdash \mathrm{M} : B}{\Gamma \vdash \lambda^T x : A.\ \mathrm{M} : \Pi^T x : A.\ B} \qquad \frac{\Gamma \vdash \mathrm{M} : \Pi^T x : B.\ A \qquad \Gamma \vdash \mathrm{N} : B}{\Gamma \vdash \mathrm{M}\ \mathrm{N} : A[N/x]}$$

$$\frac{\Gamma, x : A \vdash \mathrm{M} : \boldsymbol{el}(\tau)}{\Gamma \vdash \lambda^S x : A.\ \mathrm{M} : \boldsymbol{el}(\Pi^S x : A.\ \tau)} \qquad \frac{\Gamma \vdash \mathrm{M} : \boldsymbol{el}(\Pi^S x : A.\ \tau) \qquad \Gamma \vdash \mathrm{N} : A}{\Gamma \vdash \mathrm{M}\ \mathrm{N} : \boldsymbol{el}(\tau[N/x])}$$

$$\frac{\Gamma \vdash \mathrm{M} : A \qquad \Gamma \vdash \mathrm{N} : B[\mathrm{M}/x]}{\Gamma \vdash (\mathrm{M}, \mathrm{N})^T : \Sigma^T x : A.\ B} \qquad \frac{\Gamma \vdash \mathrm{M} : A \qquad \Gamma \vdash \mathrm{N} : \boldsymbol{el}(\tau[\mathrm{M}/x])}{\Gamma \vdash (\mathrm{M}, \mathrm{N})^S : \boldsymbol{el}(\Sigma^S x : A.\ \tau)}$$

$$\frac{\Gamma, z : \boldsymbol{el}(\Sigma^S x : A.\tau) \vdash \sigma : \boldsymbol{set} \qquad \Gamma, x : A, y : \tau \vdash \mathrm{M} : \boldsymbol{el}(\sigma[(x,y)^S/z])}{\Gamma, z : \boldsymbol{el}(\Sigma^S x : A.\ \tau) \vdash \boldsymbol{unpack}\ z\ \boldsymbol{as}\ (x, y)\ \boldsymbol{in}\ \mathrm{M} : \boldsymbol{el}(\sigma)}$$

$$\frac{\Gamma \vdash \mathrm{M} : \Sigma^T x : A.B}{\Gamma \vdash \boldsymbol{fst}\ \mathrm{M} : A} \qquad \frac{\Gamma \vdash \mathrm{M} : \Sigma^T x : A.B}{\Gamma \vdash \boldsymbol{snd}\ \mathrm{M} : B[\boldsymbol{fst}\ \mathrm{M}/x]}$$

$$\frac{\Gamma \vdash \mathrm{M} : A}{\Gamma \vdash \boldsymbol{inl}_{A,B}(\mathrm{M}) : A + B} \qquad \frac{\Gamma \vdash \mathrm{M} : B}{\Gamma \vdash \boldsymbol{inr}_{A,B}(\mathrm{M}) : A + B}$$

$$\frac{\Gamma \vdash \mathrm{M} : A + B \qquad \Gamma, x : A \vdash \mathrm{N}_1 : C \qquad \Gamma, x : B \vdash \mathrm{N}_2 : C}{\Gamma \vdash \boldsymbol{case}_{A,B}\ \mathrm{M}\ \boldsymbol{in}\ \boldsymbol{inl}(x) \Rightarrow \mathrm{N}_1 \mid \boldsymbol{inr}(x) \Rightarrow \mathrm{N}_2 : C}$$

A.9 W-types

$$\frac{\Gamma \vdash A : \boldsymbol{type} \qquad \Gamma, x : A \vdash B : \boldsymbol{type}}{\Gamma \vdash \mathcal{W}^T x : A.B : \boldsymbol{type}} \qquad \frac{\Gamma \vdash \tau : \boldsymbol{set} \qquad \Gamma, x : \boldsymbol{el}(\tau) \vdash \sigma : \boldsymbol{set}}{\Gamma \vdash \mathcal{W}^S x : \tau.\sigma : \boldsymbol{set}}$$

$$\frac{\Gamma \vdash \mathrm{M} : \Sigma^T x : A.\ (B \to \mathcal{W} x : A.B)}{\Gamma \vdash sup^T_{x:A.B}(\mathrm{M}) : \mathcal{W}^T x : A.B} \qquad \frac{\Gamma \vdash \mathrm{M} : \Sigma^T x : \boldsymbol{el}(\tau).\ (\boldsymbol{el}(\sigma) \to \boldsymbol{el}(\mathcal{W} x : \tau.\sigma))}{\Gamma \vdash sup^S_{x:\tau.\sigma}(\mathrm{M}) : \boldsymbol{el}(\mathcal{W}^S x : \tau.\sigma)}$$

$$\frac{\begin{array}{c}\Gamma \vdash A : \boldsymbol{type} \qquad \Gamma, x : A \vdash B : \boldsymbol{type} \qquad \Gamma \vdash X : \boldsymbol{type} \\ \Gamma \vdash \mathrm{M} : (\Sigma^T x : A.\ (B \to X)) \to X\end{array}}{\Gamma \vdash fold^T_{x:A.B}(\mathrm{M}) : \mathcal{W}^T x : A.B \to X}$$

$$\frac{\Gamma \vdash \tau : \boldsymbol{set} \qquad \Gamma, x : \boldsymbol{el}(\tau) \vdash \sigma : \boldsymbol{set} \qquad \Gamma \vdash X : \boldsymbol{type} \\ \Gamma \vdash \mathrm{M} : (\Sigma^T x : \boldsymbol{el}(\tau).\ (\boldsymbol{el}(\sigma) \to X)) \to X}{\Gamma \vdash fold^S_{x:\tau.\sigma}(\mathrm{M}) : \boldsymbol{el}(\mathcal{W}^S x : \tau.\sigma) \to X}$$

$$\frac{\Gamma \vdash A : \boldsymbol{type} \qquad \Gamma, x : A \vdash B : \boldsymbol{type} \qquad \Gamma \vdash X : \boldsymbol{type} \\ \Gamma \vdash \mathrm{N}_1 : (\Sigma^T x : A.\ (B \to X)) \to X \qquad \Gamma \vdash \mathrm{N}_2 : (\mathcal{W}^T x : A.B) \to X}{\Gamma \vdash (\forall y : (\Sigma^T x : A.\ (B \to \mathcal{W}^T x : A.B)).\ \mathrm{N}_2(sup^T_{x:A.B}(y)) = N_1(\pi_1(y), fold^T_{x:A:B}(N_1) \circ \pi_2(y))) \Rightarrow \mathrm{N}_2 = fold^T_{x:A.B}(\mathrm{N}_1)}$$

$$\frac{\Gamma \vdash \tau : \boldsymbol{set} \qquad \Gamma, x : \boldsymbol{el}(\tau) \vdash \sigma : \boldsymbol{set} \qquad \Gamma \vdash X : \boldsymbol{type} \\ \Gamma \vdash \mathrm{N}_1 : (\Sigma^T x : \boldsymbol{el}(\tau).\ (\boldsymbol{el}(\sigma) \to X)) \to X \qquad \Gamma \vdash \mathrm{N}_2 : (\mathcal{W}^S x : \tau.\sigma) \to X}{\Gamma \vdash (\forall y : (\Sigma^T x : \boldsymbol{el}(\tau).\ (\boldsymbol{el}(\sigma) \to \mathcal{W}^S x : \tau.\sigma)).\ \mathrm{N}_2(sup^S_{x:\tau.\sigma}(y)) = N_1(\pi_1(y), fold^S_{x:\tau.\sigma}(N_1) \circ \pi_2(y))) \Rightarrow \mathrm{N}_2 = fold^S_{x:\tau.\sigma}(\mathrm{N}_1)}$$

A.10 Heaps

For each term M : A below, iHTT contains a typing rule,

$$\frac{}{\Gamma \vdash \mathrm{M} : A}$$

$$\begin{aligned} \boldsymbol{empty} &: \boldsymbol{heap} \\ \boldsymbol{upd} &: \Pi^T \tau : \boldsymbol{set}.\ \boldsymbol{heap} \to \boldsymbol{el}(\boldsymbol{nat}) \to \boldsymbol{el}(\tau) \to \boldsymbol{heap} \\ \boldsymbol{free} &: \boldsymbol{heap} \to \boldsymbol{el}(\boldsymbol{nat}) \to \boldsymbol{heap} \\ \boldsymbol{max} &: \boldsymbol{heap} \to \boldsymbol{el}(\boldsymbol{nat}) \\ \boldsymbol{dom} &: \boldsymbol{heap} \to \boldsymbol{el}(\boldsymbol{nat}) \to \boldsymbol{bool} \\ \boldsymbol{peek} &: \boldsymbol{heap} \to \boldsymbol{el}(\boldsymbol{nat}) \to \boldsymbol{el}(1) + (\Sigma^T \tau : \boldsymbol{set}.\ \boldsymbol{el}(\tau)) \end{aligned}$$

A.11 Heap axioms

For each propostion P below, iHTT contains a typing rule,

$$\frac{}{\Gamma \vdash \bullet_{\mathrm{P}} : \boldsymbol{el}(\boldsymbol{prf}(\mathrm{P}))}$$

$\boldsymbol{max}\ \boldsymbol{empty} =_{\boldsymbol{el}(\boldsymbol{nat})} \boldsymbol{zero}$
$\forall \tau : \boldsymbol{set}.\ \forall h : \boldsymbol{heap}.\ \forall n : \boldsymbol{el}(\boldsymbol{nat}).\ \forall v : \boldsymbol{el}(\tau).$
$\quad \boldsymbol{max}\ (h[n \mapsto_\tau v]) =_{\boldsymbol{el}(\boldsymbol{nat})} \boldsymbol{if}\ n < \boldsymbol{max}\ h\ \boldsymbol{then}\ \boldsymbol{max}\ h\ \boldsymbol{else}\ n$

$\forall n : \boldsymbol{el}(\boldsymbol{nat}).\ \boldsymbol{peek}\ \boldsymbol{empty}\ n = \boldsymbol{inl}\ ()$
$\forall \tau : \boldsymbol{set}.\ \forall h : \boldsymbol{heap}.\ \forall n : \boldsymbol{el}(\boldsymbol{nat}).\ \forall v : \boldsymbol{el}(\tau).\ \forall m : \boldsymbol{el}(\boldsymbol{nat}).$
$\quad \boldsymbol{peek}\ (h[n \mapsto_\tau v])\ m = \boldsymbol{if}\ n == m\ \boldsymbol{then}\ \boldsymbol{inr}\ Pair_{\tau:\boldsymbol{set}.\boldsymbol{el}(\tau)}(\tau, v)\ \boldsymbol{else}\ \boldsymbol{peek}\ h\ m$
$\forall h : \boldsymbol{heap}.\ \forall n, m : \boldsymbol{el}(\boldsymbol{nat}).$
$\quad \boldsymbol{peek}\ (h \setminus n)\ m = \boldsymbol{if}\ n == m\ \boldsymbol{then}\ \boldsymbol{inl}\ ()\ \boldsymbol{else}\ \boldsymbol{peek}\ h\ m$

$\forall n : \boldsymbol{el}(\boldsymbol{nat}).\ \boldsymbol{dom}\ \boldsymbol{empty}\ n = \boldsymbol{false}$
$\forall \tau : \boldsymbol{set}.\ \forall h : \boldsymbol{heap}.\ \forall n : \boldsymbol{el}(\boldsymbol{nat}).\ \forall v : \boldsymbol{el}(\tau).\ \forall m : \boldsymbol{el}(\boldsymbol{nat}).$
$\quad \boldsymbol{dom}\ (h[n \mapsto_\tau v])\ m = \boldsymbol{if}\ n == m\ \boldsymbol{then}\ \boldsymbol{true}\ \boldsymbol{else}\ \boldsymbol{dom}\ h\ m$
$\forall h : \boldsymbol{heap}.\ \forall nm : \boldsymbol{el}(\boldsymbol{nat}).$
$\quad \boldsymbol{dom}\ (h \setminus n)\ m = \boldsymbol{if}\ n == m\ \boldsymbol{then}\ \boldsymbol{false}\ \boldsymbol{else}\ \boldsymbol{dom}\ h\ m$

$\forall h : \boldsymbol{heap}.\ \forall \alpha, \beta : \boldsymbol{set}.\ \forall n, m : \boldsymbol{el}(\boldsymbol{nat}).\ \forall a : \boldsymbol{el}(\alpha).\ \forall b : \boldsymbol{el}(\beta).$
$\quad h[n \mapsto_\alpha a][m \mapsto_\beta b] = \boldsymbol{if}\ n == m\ \boldsymbol{then}\ h[m \mapsto_\beta b]\ \boldsymbol{else}\ h[m \mapsto_\beta b][n \mapsto_\alpha a]$

$\forall h_1, h_2 : \boldsymbol{heap}.\ (\forall n : \boldsymbol{el}(\boldsymbol{nat}).\ \boldsymbol{peek}\ h_1\ n = \boldsymbol{peek}\ h_2\ n) \Rightarrow h_1 =_{\boldsymbol{heap}} h_2$
$\forall \mathrm{P} : \boldsymbol{heap} \to \boldsymbol{prop}.\ (\forall h : \boldsymbol{heap}.\ \forall \alpha : \boldsymbol{set}.\ \forall n : \boldsymbol{el}(\boldsymbol{nat}).\ \forall v : \boldsymbol{el}(\alpha).\ \mathrm{P}\ h \Rightarrow \boldsymbol{max}\ h < n \Rightarrow \mathrm{P}\ (h[n \mapsto_\alpha v])) \Rightarrow$
$\quad \mathrm{P}\ \boldsymbol{empty} \Rightarrow \forall h : \boldsymbol{heap}.\ \mathrm{P}\ h$

where $h[n \mapsto_\tau v]$ is shorthand for $\boldsymbol{upd}\ \tau\ h\ n\ v$ and $h \setminus n$ is shorthand for $\boldsymbol{free}\ h\ n$.

A.12 Computations

For each term M : τ below, iHTT contains a typing rule,

$$\frac{}{\Gamma \vdash \mathrm{M} : \boldsymbol{el}(\tau)}$$

$$\begin{aligned}
\boldsymbol{st} &: \Pi^S \tau : \boldsymbol{set}.\ spec\ \tau \to \boldsymbol{set}\\
\boldsymbol{ret} &: \Pi^S \tau : \boldsymbol{set}.\ \Pi^S v : \boldsymbol{el}(\tau).\ \{\lambda\_.\top\}\tau\{\lambda r, h_i, h_t.\ h_i = h_t \wedge r = v\}\\
\boldsymbol{read} &: \Pi^S \tau : \boldsymbol{set}.\ \Pi^S l : \boldsymbol{el}(\boldsymbol{nat}).\ \{\lambda h.\ \boldsymbol{dom}\ h\ l = \boldsymbol{true}\}\tau\{\lambda r, h_i, h_t.\ h_i = h_t \wedge \boldsymbol{peek}\ h_t\ l = inr(\tau, r)\}\\
\boldsymbol{write} &: \Pi^S \tau : \boldsymbol{set}.\ \Pi^S l : \boldsymbol{el}(\boldsymbol{nat}).\ \Pi^S v : \boldsymbol{el}(\tau).\ \{\lambda h.\ \boldsymbol{dom}\ h\ n = \boldsymbol{true}\}\tau\{\lambda r, h_i, h_t.\ h_t = h_i[l \mapsto_\tau v]\}\\
\boldsymbol{alloc} &: \Pi^S \tau : \boldsymbol{set}.\ \Pi^S v : \boldsymbol{el}(\tau).\ \{\lambda\_.\top\}\boldsymbol{nat}\{\lambda r, h_i, h_t.\ h_t = h_i[r \mapsto_\tau v] \wedge r \neq 0 \wedge \boldsymbol{dom}\ h_i\ r = \boldsymbol{false}\}\\
\boldsymbol{dealloc} &: \Pi^S l : \boldsymbol{el}(\boldsymbol{nat}).\ \{\lambda h.\ \boldsymbol{dom}\ h\ l = \boldsymbol{true}\}1\{\lambda\_, h_i, h_t.\ h_t = h_i \setminus l\}
\end{aligned}$$

$$
\begin{aligned}
&\boldsymbol{do} : \Pi^S \tau : \boldsymbol{set}.\ \Pi^S s_1, s_2 : spec\ \tau.\ \Pi^S \boldsymbol{el}(\boldsymbol{prf}(conseq\ s_1, s_2)).\ \boldsymbol{el}(\boldsymbol{st}\ \tau\ s_1) \to \boldsymbol{el}(\boldsymbol{st}\ \tau\ s_2)\\
&\boldsymbol{bind} : \Pi^S \tau, \sigma : \boldsymbol{set}.\ \Pi^S s_1 : spec\ \tau.\ \Pi^S s_2 : \boldsymbol{el}(\tau) \to spec\ \sigma.\ \boldsymbol{el}(\boldsymbol{st}\ \tau\ s_1) \to (\Pi^S v : \tau.\ \boldsymbol{st}_\sigma\ (s_2\ v)) \to\\
&\qquad \{\lambda h_i.\ \pi_1(s_1)\ h_i \wedge \forall x h_t.\ \pi_2(s_1)\ x\ h_i\ h_t \Rightarrow \pi_1(s_2\ x)\ h_m\}\\
&\qquad\quad \sigma\\
&\qquad \{\lambda r, h_i, h_t.\ \exists x, h.\ \pi_2(s_1)\ x\ h_i\ h_t \wedge \pi_2(s_2)\ r\ h\ h_t\}\\
&\boldsymbol{fix}_A : \Pi^S \tau : \boldsymbol{set}.\ \Pi^S s : spec\ \tau.\ ((\Pi^S x : A.\ \boldsymbol{st}\ \tau\ s) \to (\Pi^S x : A.\ \boldsymbol{st}\ \tau\ s)) \to \Pi^S x : A.\ \boldsymbol{st}\ \tau\ s
\end{aligned}
$$

where

$$spec = \lambda \tau : \boldsymbol{set}.\ (\boldsymbol{heap} \to \boldsymbol{prop}) \times (\tau \to \boldsymbol{heap} \to \boldsymbol{heap} \to \boldsymbol{prop})$$

and $\{\mathrm{P}\}\tau\{\mathrm{Q}\}$ is shorthand for $\boldsymbol{st}\ \tau\ (\mathrm{P}, \mathrm{Q})^T$.

A.13 Term definitional equality

$\boxed{\boldsymbol{\Gamma \vdash \mathrm{M} = \mathrm{N} : A}}$

$$\frac{\Gamma \vdash M : A}{\Gamma \vdash M = M : A} \qquad \frac{\Gamma \vdash M = N : A}{\Gamma \vdash N = M : A} \qquad \frac{\Gamma \vdash M_1 = M_2 : A \qquad \Gamma \vdash M_2 = M_3 : A}{\Gamma \vdash M_1 = M_3 : A}$$

$$\frac{\Gamma, x : A \vdash \mathrm{M} : B \qquad \Gamma \vdash \mathrm{N} : A \qquad U \in \{S, T, P\}}{\Gamma \vdash (\lambda^U x : A.\ \mathrm{M})\ \mathrm{N} = \mathrm{M}[\mathrm{N}/x] : B[\mathrm{M}/x]}$$

$$\frac{\Gamma \vdash \mathrm{M} : A \qquad \Gamma \vdash \mathrm{N} : B[\mathrm{M}/x]}{\Gamma \vdash \boldsymbol{fst}\ (\mathrm{M}, \mathrm{N})^T = \mathrm{M} : A} \qquad \frac{\Gamma \vdash \mathrm{M} : A \qquad \Gamma \vdash \mathrm{N} : B[\mathrm{M}/x]}{\Gamma \vdash \boldsymbol{snd}\ (\mathrm{M}, \mathrm{N})^T = \mathrm{N} : B[\mathrm{M}/x]}$$

$$\frac{\Gamma \vdash \mathrm{M} : A \qquad \Gamma \vdash \mathrm{N} : \boldsymbol{el}(\tau[\mathrm{M}/x]) \qquad \Gamma, x : A, y : \tau \vdash \mathrm{K} : \boldsymbol{el}(\sigma)}{\Gamma \vdash \boldsymbol{unpack}\ (\mathrm{M}, \mathrm{N})^S\ \boldsymbol{as}\ (x, y)\ \boldsymbol{in}\ K = K[\mathrm{M}/x, \mathrm{N}/y] : \boldsymbol{el}(\sigma)}$$

$$\frac{\Gamma \vdash \mathrm{M} : A \qquad \Gamma, x : A \vdash \mathrm{N}_1 : C \qquad \Gamma, x : B \vdash \mathrm{N}_2 : C}{\Gamma \vdash \boldsymbol{case}\ \boldsymbol{inl}(\mathrm{M})\ \boldsymbol{in}\ \boldsymbol{inl}(x) \Rightarrow \mathrm{N}_1 \mid \boldsymbol{inr}(x) \Rightarrow \mathrm{N}_2 = \mathrm{N}_1[\mathrm{M}/X] : C}$$

$$\frac{\Gamma \vdash \mathrm{M} : B \qquad \Gamma, x : A \vdash \mathrm{N}_1 : C \qquad \Gamma, x : B \vdash \mathrm{N}_2 : C}{\Gamma \vdash \boldsymbol{case}\ \boldsymbol{inr}(\mathrm{M})\ \boldsymbol{in}\ \boldsymbol{inl}(x) \Rightarrow \mathrm{N}_1 \mid \boldsymbol{inr}(x) \Rightarrow \mathrm{N}_2 = \mathrm{N}_2[\mathrm{M}/X] : C}$$

$$\frac{\Gamma \vdash \mathrm{M} : \Sigma^T x : A.\ (B \to \mathcal{W}^T x : A.B) \qquad \Gamma \vdash \mathrm{N} : (\Sigma^T x : A.\ (B \to X)) \to X}{\Gamma \vdash fold^T_{x:A.B}(\mathrm{N})(sup^T_{x:A.B}(\mathrm{M})) = \mathrm{N}(\pi_1(\mathrm{M}), fold^T_{x:A.B}(\mathrm{N}) \circ \pi_2(\mathrm{M})) : X}$$

$$\frac{\Gamma \vdash \mathrm{M} : \Sigma^T x : \boldsymbol{el}(\tau).\ (\boldsymbol{el}(\sigma) \to \boldsymbol{el}(\mathcal{W}^S a : \tau.\sigma)) \qquad \Gamma \vdash \mathrm{N} : (\Sigma^T x : \boldsymbol{el}(\tau).\ (\boldsymbol{el}(\sigma) \to X)) \to X}{\Gamma \vdash fold^S_{x:\tau.\sigma}(\mathrm{N})(sup^S_{x:\tau.\sigma}(\mathrm{M})) = \mathrm{N}(\pi_1(\mathrm{M}), fold^S_{x:\tau.\sigma}(\mathrm{N}) \circ \pi_2(\mathrm{M})) : X}$$

A.14 Type definitional equality

$\boxed{\boldsymbol{\Gamma \vdash A = B : type}}$

$$\frac{\Gamma \vdash A : \boldsymbol{type}}{\Gamma \vdash A = A : \boldsymbol{type}} \qquad \frac{\Gamma \vdash A = B : \boldsymbol{type}}{\Gamma \vdash B = A : \boldsymbol{type}} \qquad \frac{\Gamma \vdash A = B : \boldsymbol{type} \qquad \Gamma \vdash B = C : \boldsymbol{type}}{\Gamma \vdash A = C : \boldsymbol{type}}$$

$$\frac{\Gamma \vdash A = A' : \boldsymbol{type} \qquad \Gamma, x : A \vdash B = B' : \boldsymbol{type}}{\Gamma \vdash \Pi^T x : A.B = \Pi^T x : A'.B' : \boldsymbol{type}} \qquad \frac{\Gamma \vdash A = A' : \boldsymbol{type} \qquad \Gamma, x : A \vdash B = B' : \boldsymbol{type}}{\Gamma \vdash \Sigma^T x : A.B = \Sigma^T x : A'.B' : \boldsymbol{type}}$$

$$\frac{\Gamma \vdash A = A' : \boldsymbol{type} \qquad \Gamma, x : A \vdash B = B' : \boldsymbol{type}}{\Gamma \vdash \mathcal{W}^T x : A.B = \mathcal{W}^T x : A'.B'} \qquad \frac{\Gamma \vdash A = A' : \boldsymbol{type} \qquad \Gamma \vdash B = B' : \boldsymbol{type}}{\Gamma \vdash A + B = A' + B' : \boldsymbol{type}}$$

$$\frac{\Gamma \vdash \tau = \sigma : \boldsymbol{set}}{\Gamma \vdash \boldsymbol{el}(\tau) = \boldsymbol{el}(\sigma) : \boldsymbol{type}}$$

B Interpretation

This appendix contains the concrete interpretation of the underlying dependent type theory. The interpretation of terms of small types is defined in terms of a realizer, which uniquely determines a morphism in the appropriate fibre category.

$$\begin{aligned}
[\![-]\!]^{\mathrm{Ctx}} &: \mathrm{Ctx} \rightharpoonup \mathit{obj}(\mathrm{Asm}(\mathbb{V}))\\
[\![-]\!]^{\mathrm{Type}} &: \mathrm{Ctx} \times \mathrm{Term} \rightharpoonup \mathit{obj}(\mathrm{UFam}(\mathrm{Asm}(\mathbb{V})))\\
[\![-]\!]^{\mathrm{Term}} &: \mathrm{Ctx} \times \mathrm{Term} \rightharpoonup \mathit{hom}(\mathrm{UFam}(\mathrm{Asm}(\mathbb{V})))
\end{aligned}$$

B.1 Contexts $\boxed{\boldsymbol{\Gamma}}$

$$\begin{aligned}
[\![\varepsilon]\!]^{\mathrm{Ctx}} &= 1\\
[\![\Gamma, x : A]\!]^{\mathrm{Ctx}} &= (\Sigma_{i\in I} X_i, (i,x) \mapsto \{(a,b) \mid a \in E(i) \wedge b \in E_{X_i}(x)\})
\end{aligned}$$

for $(I,E) = [\![\Gamma]\!]^{\mathrm{Ctx}}$ and $(X_i, E_{X_i}) = [\![\Gamma \vdash A]\!]^{\mathrm{Type}}$.

B.2 Types $\boxed{\boldsymbol{\Gamma \vdash A : type}}$

$$\begin{aligned}
[\![\Gamma \vdash \boldsymbol{set}]\!]_i^{\mathrm{Type}} &= \nabla(\mathrm{CMPer}(\mathbb{V}))\\
[\![\Gamma \vdash \boldsymbol{prop}]\!]_i^{\mathrm{Type}} &= \nabla(2)\\
[\![\Gamma \vdash \boldsymbol{el}(\tau)]\!]_i^{\mathrm{Type}} &= \mathcal{I}_{el}([\![\Gamma \vdash \tau]\!]_i^{\mathrm{Term}})\\
[\![\Gamma \vdash \Pi^T x : A.B]\!]_i^{\mathrm{Type}} &= \left(\left\{ f \in \prod_{i\in X_i} Y_{(i,x)} \mid E_{\prod_i}(f) \neq \emptyset \right\}, E_{\prod_i}\right)\\
[\![\Gamma \vdash \Sigma^T x : A.B]\!]_i^{\mathrm{Type}} &= \left(\coprod_{x\in X_i} Y_{(i,x)}, E_{\coprod_i}\right)
\end{aligned}$$

for $(I,E) = [\![\Gamma]\!]^{\mathrm{Ctx}}$, $(X_i, E_{X_i}) = [\![\Gamma \vdash A]\!]^{\mathrm{Type}}$, and $(Y_{(i,x)}, E_{Y_{(i,x)}}) = [\![\Gamma, x : A \vdash B]\!]^{\mathrm{Type}}$, where

$$\begin{aligned}
E_{\prod_i}(f) &= \{in_{\rightarrow}(\alpha) \mid \forall x \in X_i.\ \forall e_x \in E_{X_i}(x).\ \alpha \cdot e_x \in E_{Y_{(i,x)}}(f(x))\}\\
E_{\coprod_i}(x,y) &= \{in_{\times}(\alpha,\beta) \mid \alpha \in E_{X_i}(x) \wedge \beta \in E_{Y_{(i,x)}}(y)\}
\end{aligned}$$

B.3 Small types $\boxed{\boldsymbol{\Gamma \vdash \tau : set}}$

$$\begin{aligned}
[\![\Gamma \vdash \boldsymbol{prf}(\mathrm{P})]\!]_i^{\mathrm{Term}} &= \begin{cases} \mathbb{V} \times \mathbb{V} & \text{if } [\![\Gamma \vdash \mathrm{P}]\!]_i^{\mathrm{Term}} = \top \\ \emptyset & \text{if } [\![\Gamma \vdash \mathrm{P}]\!]_i^{\mathrm{Term}} = \bot \end{cases}\\
[\![\Gamma \vdash \Pi^S x : A.\tau]\!]_i^{\mathrm{Term}} &= \left\{(a,b) \mid \forall x \in X_i.\ \forall e_1, e_2 \in E_{X_i}(x).\ (a \cdot e_1, b \cdot e_2) \in R_{(i,x)}\right\}\\
[\![\Gamma \vdash \Sigma^S x : A.\tau]\!]_i^{\mathrm{Term}} &= \overline{\left\{(a,b) \mid \exists x, y \in \coprod_{x\in X_i} [R_{(i,x)}].\ a \in E_{\coprod_i}(x) \wedge b \in E_{\coprod_i}(y) \wedge x \sim_i y\right\}}
\end{aligned}$$

for $(I,E) = [\![\Gamma]\!]^{\mathrm{Ctx}}$, $(X_i, E_{X_i}) = [\![\Gamma \vdash A]\!]^{\mathrm{Type}}$, and $R_{(i,x)} = [\![\Gamma, x : A \vdash \tau]\!]^{\mathrm{Term}}$, where

$$\begin{aligned}
E_{\coprod_i}(x,y) &= \{in_{\times}(\alpha,\beta) \mid \alpha \in E_{X_i}(x) \wedge \beta \in y\}\\
x \smile_i y \quad &\text{iff} \quad E_{\coprod_i}(x) \cap E_{\coprod_i}(y) \neq \emptyset
\end{aligned}$$

$\sim_i$ is the transitive closure of $\smile_i$, and $\overline{(-)}$ denotes monotone completion.

B.4 Terms

$\boxed{\boldsymbol{\Gamma \vdash M : A}}$

$$
\begin{aligned}
[\![\Gamma, x : A \vdash x]\!]^{\mathrm{Term}}_{(i,x)} &= x \\
[\![\Gamma \vdash \mathit{Abs}^{T}_{x:A.B}(\mathrm{M})]\!]^{\mathrm{Term}}_{i} &= x \in X_i \mapsto [\![\Gamma, x : A \vdash \mathrm{M}]\!]^{\mathrm{Term}}_{(i,x)} \\
[\![\Gamma \vdash \mathit{App}^{T}_{x:A.B}(\mathrm{M},\mathrm{N})]\!]^{\mathrm{Term}}_{i} &= [\![\Gamma \vdash \mathrm{M}]\!]^{\mathrm{Term}}_{i}([\![\Gamma \vdash \mathrm{N}]\!]^{\mathrm{Term}}_{i}) \\
[\![\Gamma \vdash \mathit{Pair}^{T}_{x:A.B}(\mathrm{M},\mathrm{N})]\!]^{\mathrm{Term}}_{i} &= ([\![\Gamma \vdash \mathrm{M}]\!]^{\mathrm{Term}}_{i}, [\![\Gamma, x : A \vdash \mathrm{N}]\!]^{\mathrm{Term}}_{(i,[\![\Gamma \vdash \mathrm{M}]\!]^{\mathrm{Term}}_{i})}) \\
[\![\Gamma \vdash \pi^{1}_{x:A.B}(\mathrm{M})]\!]^{\mathrm{Term}}_{i} &= \pi_1([\![\Gamma \vdash \mathrm{M}]\!]^{\mathrm{Term}}_{i}) \\
[\![\Gamma \vdash \pi^{2}_{x:A.B}(\mathrm{M})]\!]^{\mathrm{Term}}_{i} &= \pi_2([\![\Gamma \vdash \mathrm{M}]\!]^{\mathrm{Term}}_{i})
\end{aligned}
$$

$$
\begin{aligned}
[\![\Gamma \vdash \mathit{Abs}^{S}_{x:A.\tau}(\mathrm{M})]\!]^{\mathrm{Term}} &= \mathit{in}_{\to}(\lambda e.\mathit{in}_{\to}(\lambda x.\ m(\mathit{in}_{\times}(e,x)))) \\
[\![\Gamma \vdash \mathit{App}^{S}_{x:A.\tau}(\mathrm{M},\mathrm{N})]\!]^{\mathrm{Term}} &= \mathit{in}_{\to}\left(\lambda e.\ \begin{array}{ll} \multicolumn{2}{c}{\textbf{\textit{case}}\ m(e)\ \textbf{\textit{of}}} \\ \mathit{in}_{\to}(f) & \Rightarrow f(n(e)) \\ \text{otherwise} & \Rightarrow \bot_{\mathbb{V}} \end{array}\right) \\
[\![\Gamma \vdash \mathit{Pair}^{S}_{x:A.\tau}(\mathrm{M},\mathrm{N})]\!]^{\mathrm{Term}} &= \mathit{in}_{\to}(\lambda e.\ \mathit{in}_{\times}(m(e), n(\mathit{in}_{\times}(e, m(e))))) \\
[\![\Gamma \vdash \textbf{\textit{unpack}}\ \mathrm{M}\ \textbf{\textit{as}}\ (x,y)\ \textbf{\textit{in}}\ \mathrm{N}]\!]^{\mathrm{Term}} &= \mathit{in}_{\to}\left(\lambda e.\ \begin{array}{ll} \multicolumn{2}{c}{\textbf{\textit{case}}\ m(e)\ \textbf{\textit{of}}} \\ \mathit{in}_{\times}(a,b) & \Rightarrow n(\mathit{in}_{\times}(\mathit{in}_{\times}(e,a),b)) \\ \text{otherwise} & \Rightarrow \bot_{\mathbb{V}} \end{array}\right)
\end{aligned}
$$

where $m = [\![\Gamma \vdash M]\!]^{\mathrm{Term}}$ and $n = [\![\Gamma \vdash N]\!]^{\mathrm{Term}}$.

B.5 Base types and terms

$$
\begin{aligned}
[\![\Gamma \vdash 1]\!]^{\mathrm{Term}}_{i} &= \{(\mathit{in}_1(*), \mathit{in}_1(*))\} \\
[\![\Gamma \vdash ()]\!]^{\mathrm{Term}} &= \mathit{in}_{\to}(\lambda e.\ \mathit{in}_1(*)) \\
[\![\Gamma \vdash \textbf{\textit{bool}}]\!]^{\mathrm{Term}}_{i} &= \{(\mathit{in}_{\mathbb{N}}(1), \mathit{in}_{\mathbb{N}}(1)), (\mathit{in}_{\mathbb{N}}(2), \mathit{in}_{\mathbb{N}}(2))\} \\
[\![\Gamma \vdash \textbf{\textit{nat}}]\!]^{\mathrm{Term}}_{i} &= \{(\mathit{in}_{\mathbb{N}}(n), \mathit{in}_{\mathbb{N}}(n)) \mid n \in \mathbb{N}\} \\
[\![\Gamma \vdash \textbf{\textit{if}}\ \mathrm{M}\ \textbf{\textit{then}}\ \mathrm{N}_1\ \textbf{\textit{else}}\ \mathrm{N}_2]\!]^{\mathrm{Term}}_{i} &= \begin{cases} [\![\Gamma \vdash \mathrm{N}_1]\!]^{\mathrm{Term}}_{i} & \text{if } [\![\Gamma \vdash \mathrm{M}]\!]^{\mathrm{Term}}_{i} = [\mathit{in}_{\mathbb{N}}(1)] \\ [\![\Gamma \vdash \mathrm{N}_2]\!]^{\mathrm{Term}}_{i} & \text{if } [\![\Gamma \vdash \mathrm{M}]\!]^{\mathrm{Term}}_{i} = [\mathit{in}_{\mathbb{N}}(2)] \end{cases} \\
[\![\Gamma \vdash \textbf{\textit{true}}]\!]^{\mathrm{Term}} &= \mathit{in}_{\to}(\lambda e.\ \mathit{in}_{\mathbb{N}}(1)) \\
[\![\Gamma \vdash \textbf{\textit{false}}]\!]^{\mathrm{Term}} &= \mathit{in}_{\to}(\lambda e.\ \mathit{in}_{\mathbb{N}}(2)) \\
[\![\Gamma \vdash \textbf{\textit{zero}}]\!]^{\mathrm{Term}} &= \mathit{in}_{\to}(\lambda e.\ \mathit{in}_{\mathbb{N}}(0)) \\
[\![\Gamma \vdash \textbf{\textit{succ}}\ \mathrm{M}]\!]^{\mathrm{Term}} &= \mathit{in}_{\to}\left(\lambda e.\ \begin{array}{ll} \multicolumn{2}{c}{\textbf{\textit{case}}\ m(e)\ \textbf{\textit{of}}} \\ \mathit{in}_{\mathbb{N}}(n) & \Rightarrow \mathit{in}_{\mathbb{N}}(n+1) \\ \text{otherwise} & \Rightarrow \bot_{\mathbb{V}} \end{array}\right)
\end{aligned}
$$

C Proofs

C.1 Recursion

Definition 10. *Given $X \in \mathrm{Asm}(\mathbb{V})$ and $R \in \mathrm{UFam}(\mathrm{CMPer}(\mathbb{V}))_X$, let $\Pi_X(R)$ denote the complete monotone PER,*

$$\Pi_X(R) = \{(\alpha, \beta) \mid \forall x \in |X|.\ \forall e_x \in E_X(x).\ (\alpha \cdot e_x, \beta \cdot e_x) \in R_x\}$$

Definition 11. *Let $u : \mathbb{V} \to (\mathbb{V} \to T(\mathbb{V})) \to (\mathbb{V} \to T(\mathbb{V}))$ denote*

$$u(x : \mathbb{V})(y : \mathbb{V} \to T(\mathbb{V}))(z : \mathbb{V}) \overset{def}{=} \begin{cases} a & \text{if } x \cdot in_{\to}(\lambda z.\ in_T(y(z))) \cdot z = in_T(a) \\ \bot & \text{otherwise} \end{cases}$$

and let lfp denote $in_{\to}(\lambda e.\ \sqcup_n\, in_{\to}(\lambda z.\ in_T((u(e))^n)(\bot_{\mathbb{V}\to T(\mathbb{V})})(z)))$.

Lemma 6. *Let $X \in \mathrm{Asm}(\mathbb{V})$ and $R \in \mathrm{UFam}(\mathrm{AdmPer}(T(\mathbb{V})))$, then*

$$lfp : |(\Pi_X(in_T(R)) \to \Pi_X(in_T(R))) \to \Pi_X(in_T(R))|$$

Proof. Assume $\alpha_1\ (\Pi_X(in_T(R)) \to \Pi_X(in_T(R)))\ \alpha_2$, then we need to show that

$$lfp \cdot \alpha_1 = \sqcup_n in_{\to}(\lambda z.\ in_T(((u(\alpha_1))^n)(\bot_{\mathbb{V}\to T(\mathbb{V})})(z)))\ \Pi_X(in_T(R))\ \sqcup_n\, in_{\to}(\lambda z.\ in_T(((u(\alpha_2))^n)(\bot_{\mathbb{V}\to T(\mathbb{V})})(z))) = lfp \cdot \alpha_2$$

which follows by chain-completeness from,

$$\forall n \in \mathbb{N}.\ in_{\to}(\lambda z.\ in_T((u(\alpha_1))^n(\bot_{\mathbb{V}\to H(\mathbb{V})})(z)))\ \Pi_X(in_T(R))\ in_{\to}(\lambda z.\ in_T((u(\alpha_2))^n(\bot_{\mathbb{V}\to H(\mathbb{V})})(z)))$$

which we prove by induction on n. The base case follows from the admissibility of R. For the inductive case, assume

$$in_{\to}(\lambda z.\ in_T((u(\alpha_1))^n(\bot_{\mathbb{V}\to H(\mathbb{V})})(z)))\ \Pi_X(in_T(R))\ in_{\to}(\lambda z.\ in_T((u(\alpha_2))^n(\bot_{\mathbb{V}\to H(\mathbb{V})})(z)))$$

Assume $x \in |X|$ and $e_1, e_2 \in E_X(x)$, then it follows by the induction hypothesis and definition of $\Pi_X(in_T(R))$ that,

$$(\alpha_1 \cdot in_{\to}(\lambda z.\ in_T((u(\alpha_1))^n(\bot_{\mathbb{V}\to H(\mathbb{V})})(z))) \cdot e_1, \alpha_2 \cdot in_{\to}(\lambda z.\ in_T((u(\alpha_2))^n(\bot_{\mathbb{V}\to H(\mathbb{V})})(z))) \cdot e_2) \in in_T(R)$$

and thus,

$$(in_{\to}(\lambda z.\ in_T((u(\alpha_1))^n(\bot)(z))) \cdot e_1, in_{\to}(\lambda z.\ in_T((u(\alpha_2))^n(\bot)(z))) \cdot e_2) \in in_T(R)$$

as

$$\begin{aligned} in_{\to}(\lambda z.\ in_t((u(\alpha_i))^{n+1}(\bot(z)))) \cdot e_i &= in_T((u(\alpha_i))((u(\alpha_i))^n(\bot_{\mathbb{V}\to H(\mathbb{V})}))(e_i)) \\ &= \alpha_i \cdot in_{\to}(\lambda z.\ in_T((u(\alpha_i))^n(\bot)(z))) \cdot e_i \end{aligned}$$

C.2 Underlying DTT

Lemma 7. *Let $S, T \in \mathrm{Per}(\mathbb{V})$. Then*

$$S \to T \subseteq S \to \overline{T} = \overline{S} \to \overline{T}$$

where $\overline{(-)}$ denotes monotone completion operator:

$$\overline{R} = \bigcap\{S \in \mathrm{CMPer}(\mathbb{V}) \mid R \subseteq S\}, \quad R \in \mathrm{Per}(\mathbb{V})$$

Proof. The $S \to T \subseteq S \to \overline{T}$ and $\overline{S} \to \overline{T} \subseteq S \to \overline{T}$ inclusions are obvious.

To show $S \to \overline{T} \subseteq \overline{S} \to \overline{T}$, assume $(\alpha_1, \alpha_2) \in S \to \overline{T}$.

Define,

$$U_i \stackrel{\text{def}}{=} \{(x, y) \mid (\alpha_i \cdot x, \alpha_i \cdot y) \in \overline{T}\}$$

U_i is clearly a PER and by assumption $S \subseteq U_i$. Furthermore, U_i is monotone and complete by continuity of α_i and monotone and completeness of $\overline{T}$. Hence, $\overline{S} \subseteq U_i$.

Define

$$U \stackrel{\text{def}}{=} \{(x, y) \in \overline{S} \mid (\alpha_1 \cdot x, \alpha_2 \cdot y) \in \overline{T}\}$$

Since $\overline{S} \subseteq U_i$, U is indeed a PER and by assumption $S \subseteq U$. Again, it follows that U is monotone and complete from the continuity of α_i , monotone and completeness of $\overline{T}$ and that $U_i \subseteq \overline{T}$. Hence, $U = \overline{S}$ and thus $(\alpha_1, \alpha_2) \in \overline{S} \to \overline{T}$.

Lemma 8. *Let* $S \in \mathrm{CMPer}(\mathbb{V})$ *and* $T : \mathbb{V}/S \to \mathrm{CMPer}(\mathbb{V})$*, then*

$$\Sigma_X(Y) = \{(in_\times(a_1, b_1), in_\times(a_2, b_2)) \mid a_1 \ S \ a_2 \wedge b_1 \ T([a_1]_S) \ b_2\} \in \mathrm{CMPer}(\mathbb{V})$$

Proof. $\Sigma_S(T)$ is clearly a PER.

To show that $\Sigma_S(T)$ is monotone, assume $(a_1, b_1), (a_2, b_2) \in |\Sigma_S(T)|$, $a_1 \leq a_2$, and $b_1 \leq b_2$. Then $a_1 \ S \ a_2$ and $b_1 \ T([a_1]_S) \ b_2$ and hence $((a_1, b_1), (a_2, b_2)) \in \Sigma_S(T)$.

To show that $\Sigma_S(T)$ is complete, assume $a_1 \leq a_2 \leq \cdots$, $b_1 \leq b_2 \leq \cdots$, and $(a_i, b_i) \in |\Sigma_S(T)|$. Then by completeness, $\bigsqcup_n a_n \in |S|$. Hence, by monotonicity, $a_i \ S \ \bigsqcup_n a_n$ for every i and thus $b_i \in |T([\bigsqcup_n a_n]_S)|$ for every i, from which it follows that $\bigsqcup_n b_n \in |T([\bigsqcup_n a_n])|$ by completeness.

C.3 Computations

Definition 12. *Let* $R \in \mathrm{CMPer}(\mathbb{V})$*, then*

$$spec(R) \stackrel{def}{=} (\mathbb{H}_t \to 2) \times ([R] \to \mathbb{H}_t \to \mathbb{H}_t \to 2)$$

Lemma 9. *Let* $R, S \in \mathrm{CMPer}(\mathbb{V})$*,* $(p_1, q_1) \in spec(R)$*,* $s_2 \in [R] \to spec(S)$*, and*

$$\begin{aligned} m &\in |\Delta(in_T(hoare(R, p_1, q_1)))| \\ n &\in |\Pi_{\mathcal{I}(R)}(x \in [R] \mapsto in_T(\Delta(hoare(S, \pi_1(s_2(x)), \pi_2(s_2(x))))))| \end{aligned}$$

then $bind(m, n) \in hoare(S, p, q)$*, where*

$$\begin{aligned} p(h_i \in \mathbb{H}_t) &= p_1(h_i) \wedge \forall x \in [R].\ \forall h_t \in \mathbb{H}_t.\ q_1(x)(h_i)(h_t) \Rightarrow \pi_1(s_2(x))(h_t) \\ q(x \in [S])(h_i \in \mathbb{H}_t)(h_t \in \mathbb{H}_t) &= \exists y \in [R].\ \exists h \in \mathbb{H}_t.\ q_1(y)(h_i)(h) \wedge \pi_2(s_2(y))(x)(h)(h_t) \end{aligned}$$

Proof. Let $w \in \mathbb{W}$ and $h \in | \sim_w |$ such that $p([h]_w)$.

By assumption, there exists an $\alpha \in \mathbb{V}$, such that $m = in_T(\alpha)$ and $\alpha \in hoare(R, p_1, p_2)$. Since $p_1([h]_w)$ we thus have that that $\alpha(h) \neq \boldsymbol{err}$. If $\alpha(h) = \bot$ then $bind(m, n)(h) = \bot \in hoare(S, p, q)$, trivially. Hence, the only case we have to consider is if $\alpha(h) = (v, h')$. In this case $v \in |R|$ and $h' \in \overline{M_Q}$, where

$$M_Q \stackrel{\text{def}}{=} \{h' \in H(\mathbb{V}) \mid \exists w' \in \mathbb{W}.\ q_1([v]_R)([h]_w)([h']_{w'})\}$$

Since $v \in |R|$ there exists a β such $n \cdot v = in_T(\beta)$ and $\beta \in hoare(S, \pi_1(s_2([v]_R)), \pi_2(s_2([v]_R)))$. Define N as the subset of intermediate states for which the terminal state of β satisfies the post-condition:

$$N \stackrel{\text{def}}{=} \{h' \in H(\mathbb{V}) \mid \beta(h') = \bot \vee \exists v', h''.\ \beta(h') = (v', h'') \wedge v' \in |S| \wedge h'' \in \overline{M_B(v')}\}$$

$$M_B(v') \stackrel{\text{def}}{=} \{h'' \in H(\mathbb{V}) \mid \exists w'' \in \mathbb{W}.\ \pi_2(s_2([v]_R))([v']_S)([h]_w)([h'']_{w''})\}\}$$

It is thus sufficient to show that $\overline{M_Q} \subseteq N$. To that end, we show that N is chain-complete and that $M_Q \subseteq N$. Chain-completeness of N follows from the continuity of β and monotonicity of S.

To show that $M_Q \subseteq N$, assume $w' \in \mathbb{W}$ and $h' \in |\sim_{w'}|$ such that $q_1([v]_R)([h]_w)([h']_{w'})$. Since $p([h]_w)$, it follows that $\pi_1(s_2([v]_T))([h']_{w'})$ and hence $\beta(h') = \bot$ or $\beta(h') = (v', h'')$ such that $v' \in |S|$ and $h'' \in \overline{M_S}$, where

$$M_S = \{h'' \in \mathbb{H}_u \mid \exists w'' \in \mathbb{W}.\ \pi_2(s_2([v]_R))([v']_S)([h']_{w'})([h'']_{w''})\}$$

In the first case we trivially have that $h' \in N$. For the second case we have to prove that $\overline{M_S} \subseteq \overline{M_B([v']_S)}$, which follows from $M_S \subseteq M_B([v']_S)$.

Lemma 10. *Let $R \in \text{CMPer}(\mathbb{V})$, $[n] \in [in_{\mathbb{N}}(\{(n,n) \mid n \in \mathbb{N}\})]$, $e_n \in [n]$, $v \in [R]$, and $e_v \in v$, then*

$$write(e_n, e_v) = (\lambda h.\ \textbf{if } h(n) \neq \bot \textbf{ then } (in_1(*), h[e_n \mapsto e_v]) \textbf{ else err}) \in hoare(\{(in_1(*), in_1(*))\}, p, q(v))$$

where

$$p((w, \_) \in \mathbb{H}_t) \stackrel{def}{=} w(n)\downarrow$$

$$q(x \in [R])((w, [h]_w) \in \mathbb{H}_t)(h_t \in \mathbb{H}_t) \stackrel{def}{=} h_t = (w[n \mapsto R], [h[n \mapsto e_v]])$$

Proof. Let $w \in \mathbb{W}$ and $h \in |\sim_w|$ such that $p(w, [h]_w)$. Then $w(n)\downarrow$ and thus $h(e_n) \neq \bot$. Hence,

$$write(e_n, e_v)(h) = (in_1(*), h[e_n \mapsto e_v])$$

Furthermore, $in_1(*) \in |\{(in_1(*), in_1(*))\}|$ and

$$q([in_1(*)])(w, [h]_w)(w[n \mapsto R], [h[e_n \mapsto e_v]])$$

C.4 W-types

In this appendix we construct W-types in Asm($\mathbb{V}$), from the W-types in Sets. We take it as an axiom that Sets has W-types. From this axiom, we derive an induction principle and a dependent recursion principle for W-types in Sets (Lemmas 11 and 12). W-types in Asm($\mathbb{V}$) are then defined as a "hereditarily realized" subset of the underlying W-type in Sets (Definition 13). From the dependent recursion principle we derive a recursion principle on this "hereditarily realized" subset (Lemma 13), which we use to prove that the "hereditarily realized" subset is indeed a W-type in Asm($\mathbb{V}$) (Lemma 14).

Axiom 1 *Let $A \in$ Sets and $B \in \text{Fam}(\text{Sets})_A$, then there exists a set W with an isomorphism, sup : $\left(\coprod_{a \in A} B_a \to W\right) \cong W$, such that for any set $X \in$ Sets and function $g : (\coprod_{a \in A} B_a \to X) \to X$ there exists a unique function $h : W \to X$, satisfying, $\forall a \in A.\ \forall f \in B_a \to W.\ h(sup(a, f)) = g(a, h \circ f)$.*

Given $A \in$ Sets and $B \in \text{Fam}(\text{Sets})_A$, we use $W_{A,B}$ and $sup_{A,B}$ to denote a set and isomorphism satisfying the conditions of Axiom 1. Given a morphism g as in Axiom 1, we use $fold(g)$ to refer to the meadiating morphism induced by g.

Lemma 11. *Let $A \in$ Sets, $B \in \text{Fam}(\text{Sets})_A$ and $V \subseteq W_{A,B}$, such that*

$$\forall a \in A.\ \forall f \in B_a \to W_{A,B}.\ (\forall b \in B_a.\ f(b) \in V) \Rightarrow sup(a, f) \in V$$

then $V = W$.

Proof. Define $g : (\coprod_{a\in A} B_a \to V) \to V$ as the function, $g(a,f) = sup(a,f)$. Then by the uniqueness of $fold(sup)$ we have that,

$$fold(g) = fold(sup) = id_W$$

and thus $W \subseteq V$.

Lemma 12. *Let $A \in \mathrm{Sets}$, $B \in \mathrm{Fam}(\mathrm{Sets})_A$, $X \in \mathrm{Fam}(\mathrm{Sets})_W$ and*

$$g : \Pi_{a\in A}\Pi_{f\in B_a\to W}(\Pi_{b\in B_a} X_{f(b)}) \to X_{sup(a,f)}$$

there exists a unique $h : \Pi_{w\in W} X_w$, satisfying,

$$\forall a \in A.\ \forall f \in B_a \to W.\ h(sup(a,f)) = g(a,f,h\circ f)$$

Proof. Define $g' : (\Sigma_{a\in A} B_a \to \Sigma_{w\in W} X_w) \to \Sigma_{w\in W} X_w$ as follows,

$$\begin{aligned} g'(a,f) = (&sup(a, \lambda b \in B_a.\ \pi_1(f(b))),\\ &g(a, \lambda b \in B_a.\ \pi_1(f(b)), \lambda b \in B_a.\ \pi_2(f(b)))) \end{aligned}$$

and let $h' : W \to \Sigma_{w\in W} X_w$ denote the unique function satisfying,

$$\forall a \in A.\ \forall f \in B_a \to W.\ h'(sup(a,f)) = g'(a, h'\circ f) \tag{1}$$

Define

$$V = \{w \in W \mid \pi_1(h'(w)) = w\}$$

then for any $a \in A$, $f \in B_a \to w$, such that $\forall b \in B_a.\ f(b) \in V$, we have that

$$\pi_1(h'(sup(a,f))) = \pi_1(g'(a,h'\circ f)) = sup(a, \lambda b \in B_a.\ \pi_1(h'\circ f)(b)) = sup(a,f)$$

and thus $V = W$. Define $h : \Pi_{w\in W} X_w$ as,

$$h(w) = \pi_2(h'(w))$$

then

$$\begin{aligned} h(sup(a,f)) &= \pi_2(h'(sup(a,f))) = \pi_2(g'(a,h'\circ f))\\ &= g(a, \lambda b \in B_a.\ \pi_1(h'(f(b))), \lambda b \in B_a.\ \pi_2(h'(f(b))))\\ &= g(a,f,h\circ f) \end{aligned}$$

To prove uniqueness, assume $k : \Pi_{w\in W} X_w$ is another meadiating morphism. Then $k' : W \to \Sigma_{w\in W} X_w$ defined as follows,

$$k'(w) \stackrel{\text{def}}{=} (w, k(w))$$

satisfies (1):

$$\begin{aligned} k'(sup(a,f)) &= (sup(a,f), g(a,f,k\circ f))\\ &= (sup(a, \lambda b \in B_a.\ \pi_1(k'(f(b)))), g(a, \lambda b \in B_a.\ \pi_1(k'(f(b))), \lambda b \in B_a.\ \pi_2(k'(f(b)))))\\ &= g'(a, k'\circ f) \end{aligned}$$

and thus, $k' = h'$ from which it follows that $h = \pi_2 \circ h' = \pi_2 \circ k' = k$.

Definition 13. *Given $X \in \mathrm{Asm}(\mathbb{V})$ and $Y \in \mathrm{UFam}(\mathrm{Asm}(\mathbb{V}))_X$, let $\dot{W}_{X,Y}$ denote the assembly,*

$$\dot{W}_{X,Y} = (\{w \in W_{|X|,|Y|} \mid E_W(w) \neq \emptyset\}, E_W)$$

where $E_W : W_{|X|,|Y|} \to \mathrm{P}(\mathbb{V})$ denotes the unique function satisfying,

$$E_W(sup(x,f)) = \{in_\times(e_x, e_f) \mid e_x \in E_X(x) \wedge \forall b \in |Y_x|.\ \forall e_y \in E_{Y_x}(y).\ e_f \cdot e_y \in E_W(f(y))\}$$

Lemma 13. *Let $X \in \mathrm{Asm}(\mathbb{V})$, $Y \in \mathrm{UFam}(\mathrm{Asm}(\mathbb{V}))_X$, $Z \in \mathrm{Sets}$ and $g : (\coprod_{x\in|X|}(|Y_x| \to Z)) \to Z$, then there exists a unique $h : |\dot{W}_{X,Y}| \to Z \in \mathrm{Sets}$ satisfying,*

$$\forall x \in |X|.\ \forall f \in |Y_x| \to W_{|X|,|Y|}.\ E_W(sup(x,f)) \neq \emptyset \Rightarrow h(sup(x,f)) = g(x, h \circ f) \tag{2}$$

Proof. Define $Z_w = \{* \mid E_W(w) \neq \emptyset\} \to Z$ and

$$g' : \Sigma_{x\in|X|}\Sigma_{f\in|Y_x|\to W_{|X|,|Y|}}(\Pi_{y\in|Y_x|}Z_{f(b)}) \to Z_{sup(x,f)}$$

as follows,

$$g'(x \in |X|, f \in |Y_x| \to W_{|X|,|Y|}, r \in \Pi_{y\in|Y_x|}Z_{f(b)})(*) = g(x, y \in |Y_x| \mapsto r(y)(*))$$

which is well-defined, since if $E_W(sup(x,f)) \neq \emptyset$, then $E_W(f(y)) \neq \emptyset$ for every $y \in |Y_x|$. There thus exists a unique $h' : \Pi_{w\in W}Z_w$ satisfying,

$$\forall x \in |X|.\ \forall f \in |Y_x| \to W_{|X|,|Y|}.\ E_W(sup(x,f)) \neq \emptyset \Rightarrow h'(sup(x,f))(*) = g(x, y \in |Y_x| \mapsto (h' \circ f)(b)(*))$$

Lastly, define $h : |\dot{W}_{X,Y}| \to Z$ as $h(v) = h'(v)(*)$, then

$$\forall x \in |X|.\ \forall f \in |Y_x| \to W_{|X|,|Y|}.\ E_W(sup(x,f)) \neq \emptyset \Rightarrow h(sup(x,f)) = g(x, h \circ f)$$

To show uniqueness, assume $k : |\dot{W}_{X,Y}| \to Z$ satisfying (2). Then,

$$k'(w \in W_{|X|,|Y|}) \overset{\text{def}}{=} x \in \{* \mid E_W(w) \neq \emptyset\} \mapsto k(w)$$

satisfies,

$$k'(sup(x,f))(*) = k(sup(x,f)) = g(x, k \circ f) = g(x, b \in B_a \mapsto (k' \circ f)(b)(*))$$

for $x \in |X|$ and $f \in |Y_x| \to W_{|X|,|Y|}$, with $E_W(sup(x,f)) \neq \emptyset$. Hence $k' = h'$ and thus $h(v) = h'(v)(*) = k'(v)(*) = k(v)$.

Definition 14. *Given $X \in \mathrm{Asm}(\mathbb{V})$ and $Y \in \mathrm{UFam}(\mathrm{Asm}(\mathbb{V}))_X$, let $P_{X,Y} : \mathrm{Asm}(\mathbb{V}) \to \mathrm{Asm}(\mathbb{V})$ denote the functor,*

$$\begin{aligned} P_{X,Y}(A) &= (\{(x,f) \in \Sigma_{x\in|X|}|Y_x| \to |A| \mid E_\Pi(f) \neq \emptyset\}, E_P) \\ P_{X,Y}(f : A \to B) &= (x,g) \mapsto (x, f \circ g) \end{aligned}$$

where

$$\begin{aligned} E_P(x,f) &= \{(e_x, e_f) \mid e_x \in E_X(x) \wedge e_f \in E_{\Pi_x}(f)\} \\ E_{\Pi_x}(f) &= \{\alpha \mid \forall y \in |Y_x|.\ \forall e_y \in E_{Y_x}(y).\ \alpha \cdot e_y \in E_A(f(y))\} \end{aligned}$$

Lemma 14. *Let $X \in \mathrm{Asm}(\mathbb{V})$ and $Y \in \mathrm{UFam}(\mathrm{Asm}(\mathbb{V}))_X$ then*

$$sup_{|X|,|Y|} : P_{X,Y}(\dot{W}_{X,Y}) \to \dot{W}_{X,Y} \in \mathrm{Asm}(\mathbb{V})$$

realized by the identity, is an initial $P_{X,Y}$-algebra.

Proof. Given $h : P_{X,Y}(A) \to A \in \mathrm{Asm}(\mathbb{V})$, define $u' : |\dot{W}_{X,Y}| \to |A| + 1$ as the unique map satisfying,

$$u'(sup(x,f)) = \begin{cases} inl(h(x, \pi_l \circ u' \circ f)) & (\forall y \in |Y_x|.\ (\pi_l \circ u' \circ f)(y)\downarrow) \wedge E_{\Pi_x}(\pi_l \circ u' \circ f) \neq \emptyset \\ inr(*) & \text{otherwise} \end{cases}$$

where $\pi_l : |A| + 1 \rightharpoonup |A|$ is defined as follows,

$$\pi_l(x) = \begin{cases} y & \text{if } x = inl(y) \\ \boldsymbol{undef} & \text{otherwise} \end{cases}$$

Assume e_h is a h-realizer and define α and U as follows,

$$\alpha \overset{\text{def}}{=} in_{\rightarrow}(fix(\lambda e_u.\ \lambda in_{\times}(e_a, e_f).\ e_h \cdot (e_a, e_u \circ e_f)))$$
$$U \overset{\text{def}}{=} \{v \in |\dot{W}_{X,Y}| \mid \exists a \in |A|.\ u'(a) = inl(x) \wedge \forall e_v \in E_W(v).\ \alpha \cdot e_v \in E_A(a))\}$$

where fix is the least fixed point operator on $\mathbb{V} \to \mathbb{V}_\perp$.

To show that $U = |\dot{W}_{X,Y}|$, let $x \in |X|$ and $f : |Y_x| \to W_{|X|,|Y|}$ such that $E_\Pi(x, f) \neq \emptyset$ and assume that

$$\forall y \in |Y_x|.\ f(y) \in U$$

then it follows by the induction hypothesis that

$$\forall y \in |Y_x|.\ (\pi_l \circ u' \circ f)(y)\downarrow \wedge E_{\Pi_x}(\pi_l \circ u' \circ f) \neq \emptyset$$

and hence,

$$u'(sup(x, f)) = inl(h(x, \pi_l \circ u' \circ f))$$

Assume $(e_x, e_f) \in E_W(sup(x, f))$. Then $(e_x, \alpha \circ e_f) \in E_P(x, \pi_l \circ u' \circ f)$, since by the induction hypothesis, α realizes $\pi_l \circ u'$ for $w \in W_{|X|,|Y|}$ in the image of f. Hence, as e_h realizes h,

$$\alpha \cdot (e_x, e_f) = e_h \cdot (e_x, \alpha \circ e_f) \in E_A(h(x, \pi_l \circ u' \circ f))$$

and thus $sup(x, f) \in U$.

Define $u : |\dot{W}_{X,Y}| \to |A|$ as

$$u = \pi_l \circ u'$$

then

$$u(sup(a, f)) = h(a, u \circ f)$$

and u is further realized by α.

To prove uniqueness, assume $v : |\dot{W}_{X,Y}| \to |A|$ satisfying $v \circ sup = h \circ P_{X,Y}(v)$. Then $v' = inl \circ v$ satisfies

$$\forall x.\ \forall f.\ v'(sup(x, f)) = inl(h(x, \pi_l \circ v' \circ f))$$

and thus by uniqueness of u', $u' = v'$ and thus $u = v$.

Definition 15. *Let $(X, E_X) \in \text{Asm}(\mathbb{V})$ then (X, E_X) is complete monotone iff*

$$\forall x, y \in X.\ (\exists a, b \in \mathbb{V}.\ a \in E_X(x) \wedge b \in E_X(y) \wedge a \leq b) \Rightarrow x = y$$

and

$$\forall x \in X.\ \forall c : \mathrm{N} \to_m \mathbb{V}.\ (\forall n \in \mathbb{N}.\ c(n) \in E_X(x)) \Rightarrow \sqcup_n c(n) \in E_X(x)$$

Lemma 15. *Let $X \in \text{Asm}(\mathbb{V})$ and $Y \in \text{UFam}(\text{Asm}(\mathbb{V}))_X$. If X is complete monotone and Y_x is complete monotone for each $x \in |X|$, then $\dot{W}_{X,Y}$ is complete monotone.*

Proof. We prove

$$\begin{aligned}
&\forall w \in W_{|X|,|Y|}. \\
&\qquad (\forall w' \in W_{|X|,|Y|}.\ (\exists a, b.\ a \in E_W(w) \wedge b \in E_W(w') \wedge a \leq b) \Rightarrow w = w') \wedge \\
&\qquad (\forall c : \mathbb{N} \to_m \mathbb{V}.\ (\forall n \in \mathbb{N}.\ c(n) \in E_W(w)) \Rightarrow \bigsqcup_n c(n) \in E_W(w))
\end{aligned}$$

by induction on $w \in W_{|X|,|Y|}$. Assume $x_1 \in |X|$ and $f_1 \in |Y_x| \to W_{|X|,|Y|}$.

We prove the first conjunct by induction on $w' \in W_{|X|,|Y|}$. Assume $x_2 \in |X|$ and $f_2 \in |Y_x| \to W_{|X|,|Y|}$, such that there exists $(e_{x_1}, e_{f_1}) \in E_W(x_1, f_1)$ and $(e_{x_2}, e_{f_2}) \in E_W(x_2, f_2)$ with $(e_{x_1}, e_{f_1}) \leq (e_{x_2}, e_{f_2})$. By monotonicity of X we thus have that $x_1 = x_2$. Furthermore, given any $y \in |Y_x|$ and $e_{f_1} \cdot e_y \leq e_{f_2} \cdot e_y$, where $e_y \in E_{Y_x}(y)$, and thus $f_1(y) = f_2(y)$, by the induction hypothesis for w.

The second conjunct follows easily from continuity of f_1 realizers and the induction hypothesis.