The Transfinite Iris Documentation

May 14, 2021

Abstract

This document describes formally the transfinite version of the Iris program logic. It is based on the technical documentation of Iris, available at https://plv.mpi-sws.org/iris/appendix-3.3.pdf. Every result in this document has been fully verified in Coq.

Contents

1 Introduction

This document describes formally the Transfinite Iris base logic and the various program logics for safety, termination-preserving refinement, and termination defined on top, and serves as a technical documentation. It is based on the technical documentation of Iris[1].

In particular, the following sections of this document describe aspects specifically relevant to Transfinite Iris and saw major changes, while the rest has only received minor changes from the Iris technical documentation:

- Section 2 introduces step-index types, an abstraction over ordinals, and explores notions of existential properties which hold for different "sizes" of step-index types.

  In particular, this also includes a description of our formalization of ordinals (Section 2.2) and a proof of the existential property for ordinals (Lemma 4).

- In Sections 3.1 and 3.2, we define OFEs and COFEs, which are at the basis of our step-indexed model, and which saw an extension to transfinite step-indices.

- Section 4 presents Transfinite Iris' base logic, which has received a few careful updates due to the transfinite model. In particular, Section 4.3 explains the later-commmuting rules which are not validated anymore and Section 4.4 shows a new way to obtain soundnes results.

- In Section 5, we present a solution to the recursive domain equation that needs to be solved to obtain a model supporting higher-order ghost state.

- Section 9 defines three program logics on top of Transfinite Iris' base logic. The safety program logic (Section 9.1) has seen minor changes related to the placement of the later modality.

  Sections 9.2-9.5 motivate and show our all-new program logics for termination-preserving refinement and termination.

- Section 11 shows how to instantiate the termination and refinement program logics for Iris' HeapLang language.

- In Section 12, we describe how the logical relation for termination from Spies et al. [2021] is encoded and extended in our termination framework.

1.1 Meta-Logic

We use a constructive extensional type theory as a foundation for stating definitions and proving lemmas. This theory features a hierarchy of type universes $\mathbb{T}_0 \subseteq \mathbb{T}_1 \subseteq \mathbb{T}_2 \subseteq \ldots$, as well as an impredicative universe *Prop* $\subseteq \mathbb{T}_0$ of meta-level propositions. Think of `Prop` in Coq or $\mathbb{B}$ in classical mathematics. We write $\mathbb{T}$ to denote an arbitrary suitable universe of the hierarchy as the exact universes do not matter except for a few instances. The standard notation $x : X$ to denote that x is of type X is used.

We make use of the standard type formers. Given two types X, Y, the product type $X \times Y$ contains pairs (x, y) of elements $x : X$ and $y : Y$.

The sum type $X + Y ::= L(x : X) | R(y : Y)$ contains elements of X or Y, while the option type $X^? ::= S(x : X) | \bot$ adds an "error element" $\bot$ to X. For elements $z : X + Y$ of sum types, we slightly

[1] available at https://plv.mpi-sws.org/iris/appendix-3.3.pdf

abuse notation and write $z : X$ for the assertion that $z = L(x)$ for some $x : X$ and $z : Y$ for $z = R(y)$ with $y : Y$. Similarly, for elements $m : X^?$, $m : X$ denotes that $m = S(x)$ for some $x : X$.

For a function $P : X \to \mathbb{T}$, $\Sigma(x : X).P\ x$ denotes the standard dependent sum type containing dependent pairs $(x : X, y : P\ x)$. In case that $P : X \to \mathit{Prop}$, one can imagine $\Sigma(x : X).P\ x$ as the subset type of X whose objects satisfy the predicate P.

Similarly, for $P : X \to \mathbb{T}$, $\prod(x : X).P\ x$ is the dependent product type containing functions $\lambda x.e$ where the return type may depend on the argument x. In the special case that $P : X \to \mathit{Prop}$, we also write $\forall(x : X).P\ x$ to emphasize the connection to universal quantification via the Curry-Howard isomorphism.

For some results, we need non-constructive reasoning. We will always make clear in which places we need to assume classical axioms. The following axioms are of relevance here:

- *Functional Extensionality* (FE): If $f, g : X \to Y$ and $\forall x. fx = gx$, then $f = g$.
- *Propositional Extensionality* (PE): If $P, Q : \mathit{Prop}$ with $P \leftrightarrow Q$, then $P = Q$.
- *Excluded Middle* (XM): If $P : \mathit{Prop}$, then $P \vee \neg P$.
- *Choice*: For $P : X \to \mathit{Prop}$, if $\exists x.P\ x$, then $\Sigma x.P\ x$.

In several instances, we need to "quotient" some type X by an equivalence relation $\equiv$, obtaining a type $X/_{\equiv}$ with $x = y \Leftrightarrow x \equiv y$ for $x, y : X/_{\equiv}$. While this is not natively supported in most type theories, we can formally resolve this in the standard way by defining a setoid structure with $\equiv$ on top of X and overloading the usual equality sign $=$ to refer to the relation $\equiv$ in corresponding contexts. As the formal details are not interesting, we will not go into more detail on paper.

2 Step-Indices

From the perspective of the model, Iris is a particular step-indexed logical relation. Traditionally, only natural numbers are used for step-indexing. For transfinite Iris, we allow more general types to be used as step-indices such as constructive and non-constructive ordinals. As we shall see in the following, each of these types offers its own advantages and disadvantages. We allow the user of Iris the choice of which step-index type is appropriate by parameterizing definitions over the type that is used. We impose the following requirements on step-index types:

Definition 1 (Step-Index Type). *A step-index type* $(\mathcal{I}, \prec, \mathsf{S}, 0)$ *consists of a type* $\mathcal{I}$*, a relation* $\prec : \mathcal{I} \to \mathcal{I} \to$ *Prop on* $\mathcal{I}$*, a successor function* $\mathsf{S} : \mathcal{I} \to \mathcal{I}$*, and a zero element* $0 : \mathcal{I}$*, with:*

$$\prec \text{ is transitive and well-founded} \qquad \text{(INDEX-REL)}$$

$$\forall \alpha, \beta.\ (\alpha \prec \beta) + (\alpha = \beta) + (\beta \prec \alpha) \qquad \text{(INDEX-LINEAR)}$$

$$\forall \alpha.\ \neg(\alpha \prec 0) \qquad \text{(INDEX-ZERO-LEAST)}$$

$$\forall \alpha.\ \alpha \prec \mathsf{S}\,\alpha \qquad \text{(INDEX-SUCC-GREATER)}$$

$$\forall \alpha, \beta.\ \alpha \prec \beta \Rightarrow \mathsf{S}\,\alpha \preceq \beta \qquad \text{(INDEX-SUCC-LEAST-GREATER)}$$

$$\forall \alpha.\ (\Sigma \beta.\ \alpha = \mathsf{S}\,\beta) + (\forall \beta \prec \alpha.\ \mathsf{S}\,\beta \prec \alpha) \qquad \text{(INDEX-DISTINGUISH-LIMIT)}$$

The values of $\mathcal{I}$ *are called step-indices. By* $\preceq$*, we denote the reflexive closure of* $\prec$*.*

Note that the disjunctions in INDEX-LINEAR and INDEX-DISTINGUISH-LIMIT as well as the existential quantifier in INDEX-DISTINGUISH-LIMIT have to carry computational content, meaning we can write functions which inspect whether the left or right hand side was chosen and can access the witness β. The notion of well-foundedness of $\prec$ is also constructive, allowing us to define functions by well-founded recursion on the step-index.

We define the following notion of *limit indices*:

Definition 1 (Limit Indices)**.** We say that an index α is a limit index if

$$\forall \beta \prec \alpha.\ \mathsf{S}\,\beta \prec \alpha.$$

We say that α is a proper limit index if it is additionally distinct from 0.

Intuitively, no matter how often we apply S to $\beta \prec \alpha$, we can never reach α. This aligns with the notion of limit ordinals (in set-theory) when we provide an instance for ordinals below. In that case, the first limit ordinal is ω: it is the smallest ordinal that is larger than every natural number n. No matter how often we apply S to 0, we will always stay below ω.

The property INDEX-DISTINGUISH-LIMIT states that any index α is either a *successor index* (s.t. there exists β with $\alpha = \mathsf{S}\,\beta$) or a limit index. More precisely, we can derive that any index is either 0, a successor index, or a proper limit index.

We can use the constructive notion of well-foundedness to derive a principle of transfinite recursion, splitting into the three cases of zero indices, successor indices, and proper limit indices:

Lemma 1 (Transfinite Recursion)**.** *Let* $(\mathcal{I}, \prec, \mathsf{S}, 0)$ *be a step-index type and* $P : \mathcal{I} \to \mathbb{T}$*. We have the*

following recursor:

$$
\begin{aligned}
&P(0) \to \\
&(\forall \alpha.\, P(\alpha) \to P(\mathsf{S}\,\alpha)) \to \\
&(\forall \alpha.\, \alpha \textit{ is a proper limit index} \to (\forall \beta \prec \alpha.\, P(\beta)) \to P(\alpha)) \to \\
&\forall \alpha.\, P(\alpha)
\end{aligned}
$$

Proof. We apply well-founded recursion, enabled by our well-foundedness assumption on $\prec$. Thus, let an index α be given and assume the inductive hypothesis $\forall \beta \prec \alpha.\, P(\beta)$. By INDEX-LINEAR, we decide whether $\alpha = 0$. If so, we are done by assumption, otherwise, we use INDEX-DISTINGUISH-LIMIT to decide whether α is a successor index $\alpha = \mathsf{S}\,\beta$ or a limit index. In the successor case, we use the inductive hypothesis for β and the second assumption. In the limit case, we know that α must be a proper limit index and the proof is done by the third assumption and the inductive hypothesis. □

Finite Step-Index Types and Transfinite Step-Index Types

Definition 2 (Finite Step-Index Types)**.** Let $(\mathcal{I}, \prec, \mathsf{S}, 0)$ be a step-index type. We say that it is *finite* if

$$\forall \alpha.\, (\forall \beta.\, \alpha \preceq \beta) + \Sigma \beta.\, \beta \prec \alpha \wedge \forall \gamma.\, \gamma \prec \alpha \Rightarrow \gamma \preceq \beta \qquad \text{(IDX-FINITE)}$$

While this definition may look complicated, it essentially just states that every index is either the zero index or that there exists a direct predecessor (thus ruling out limit indices).

Definition 3 (Transfinite Step-Index Types)**.** Let $(\mathcal{I}, \prec, \mathsf{S}, 0)$ be a step-index type. We say that it is *transfinite* if there is an operation

$$\mathsf{jump}(\alpha) : \Sigma \beta.\, \forall n.\ \mathsf{S}^n\, \alpha \prec \beta. \qquad \text{(IDX-JUMP-LIMIT)}$$

$\mathsf{jump}(\alpha)$ yields a limit index bounding α (but not necessarily the least such limit index).

2.1 Constructive Step-Indices

Traditionally, natural numbers are used for step-indexing. In the sense of Definition 1, natural numbers form a step-index type with $(\mathbb{N}, <, n \mapsto n + 1, 0)$. Natural numbers contain all ordinals up to the first proper limit ordinal ω. In this sense, natural numbers are *finite* step-indices (according to Definition 2). They contain no proper limit index.

We obtain a *transfinite* notion of step-indices by extending the indices we consider with the ordinals $\omega, \omega + 1, \ldots$ up to but not including ω^2:

$$0, 1, \ldots, \omega, \omega + 1, \ldots, \omega \cdot 2, \ldots, \omega \cdot 3, \ldots, \omega \cdot 4, \ldots$$

Formally, each such ordinal $\omega \cdot m + n$ may be represented as a pair (m, n). For example, we represent the ordinal $\omega + 42$ by $(1, 42)$, the ordinal ω by $(1, 0)$, and the ordinal 1 by $(0, 1)$.

We obtain an ordering on the pairs that coincides with the ordering on the ordinals they represent by ordering them lexicographically:

$$(m, n) \prec (m', n') \triangleq (m < m') \vee (m = m' \wedge n < n')$$

The successor is given by $\mathsf{S}(m, n) \triangleq (m, n + 1)$ and the zero by $(0, 0)$. Each ordinal of the form $\omega \cdot (m + 1)$, represented by $(m + 1, 0)$, is a proper limit ordinal since it is larger than all of the ordinals $\omega \cdot m + n$ for $n = 0, 1, \ldots$. Thus, we can show that this forms a transfinite index type by defining $\mathsf{jump}(m, n) \triangleq (m + 1, n)$.

Lexicographic Product To obtain step-indices for larger ordinals than ω^2 such as $\omega^3, \omega^4, \ldots$ we generalize the above approach. We define operation $\mathcal{I} \times_{\text{lex}} \mathcal{J}$ which constructs the lexicographic product of two step-index types $\mathcal{I}$ and $\mathcal{J}$. We then obtain step-indices up to ω^2 as $\mathbb{N} \times_{\text{lex}} \mathbb{N}$, step-indices up to ω^3 as $\mathbb{N} \times_{\text{lex}} (\mathbb{N} \times_{\text{lex}} \mathbb{N}), \ldots$

Definition 2 (Lexicographic Step-Index Product). *Let* $(\mathcal{I}, \prec_{\mathcal{I}}, \mathsf{S}_{\mathcal{I}}, 0_{\mathcal{I}})$ *and* $(\mathcal{J}, \prec_{\mathcal{J}}, \mathsf{S}_{\mathcal{J}}, 0_{\mathcal{J}})$. *We define:*

$$\mathcal{I} \times_{lex} \mathcal{J} \triangleq (\mathcal{I} \times \mathcal{J}, \prec_{lex}, \mathsf{S}_{lex}, 0_{lex})$$

where:

$$\begin{aligned}(i, j) \prec_{lex} (i', j') &\triangleq (i \prec_{\mathcal{I}} i') \text{ or } (i = i' \text{ and } j \prec_{\mathcal{J}} j') \\ \mathsf{S}_{lex}(i, j) &\triangleq (i, \mathsf{S}_{\mathcal{J}}\, j) \\ 0_{lex} &\triangleq (0_{\mathcal{I}}, 0_{\mathcal{J}})\end{aligned}$$

2.2 Non-Constructive, Large Step-Indices

All of the constructions above are constructive, meaning they do not require any classical reasoning principles such as the law of excluded middle or the axiom of choice. If we assume such reasoning principles, we can construct even larger ordinals such as uncountable ordinals[2]. As we shall see in Section 2.3, such large ordinals can be used to derive powerful existential properties.

We (non-constructively) construct such large ordinals by embedding a set theory in our type theory.

Ordinals in Set Theory

The construction of these large ordinals in our type theory proceeds in two steps, building on the work of Kirst and Smolka [2018]. First, a type of sets is defined in type theory using Aczel trees [Aczel, 1978; Werner, 1997]. Second, the notion of ordinals is defined on top of the set theory.

Aczel Trees In the following, we construct sets from *Aczel trees*. As the name indicates, Aczel trees can be used to represent trees — finitely and infinitely branching trees, to be precise.

Definition 3 (Aczel Trees). *We inductively define Aczel trees as:*

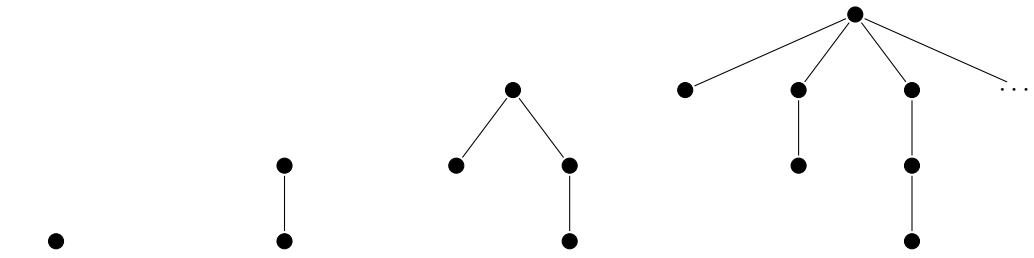
$$a, b : \mathsf{Acz} ::= \mathsf{T}(X : \mathit{Type})(f : X \to \mathsf{Acz})$$

with the projections:

$$\begin{aligned}&\pi_1 : \mathsf{Acz} \to \mathit{Type} && \pi_2 : \Pi a : \mathsf{Acz}.\, \pi_1 a \to \mathsf{Acz} \\ &\pi_1(\mathsf{T}\, X f) \triangleq X && \pi_2(\mathsf{T}\, X f) \triangleq f\end{aligned}$$

The intuition behind this definition is that each fx for $x : X$ is a direct subtree of the tree $\mathsf{T}\, X f$. For example, we obtain the empty tree, displayed in Figure 1a, as $a_0 \triangleq \mathsf{T}\, \bot\, \mathsf{abort}_{\mathsf{Acz}}$ where $\bot$ is the empty type and $\mathsf{abort}_{\mathsf{Acz}} : \bot \to \mathsf{Acz}$ is the vacuous function from the empty type to Aczel trees. We obtain a singleton tree, displayed in Figure 1b, as $a_1 \triangleq \mathsf{T}\, 1\, (() \mapsto a_0)$ and a binary tree, displayed in Figure 1c, as $a_2 \triangleq \mathsf{T}\, \mathbb{B}\, (b \mapsto \mathsf{if}\, b\, \mathsf{then}\, a_0\, \mathsf{else}\, a_1)$. We obtain an infinite tree, displayed in Figure 1d, as $a_\infty \triangleq \mathsf{T}\, \mathbb{N}\, (n \mapsto s^n a_0)$ where $sa \triangleq \mathsf{T}\, 1\, (() \mapsto a)$.

[2]Uncountable ordinals are those ordinals α where the number of ordinals $\beta \prec \alpha$ is uncountable.



(a) Empty (b) Singleton (c) Binary (d) Infinite

Figure 1: Tree Examples

Constructive Set Theory To build a set theory on top op Aczel trees, we need a suitable notion of membership and equality. For membership, we consider the direct subtrees of a tree a to be the elements of a. For equality, we incorporate the usual notion of equality on sets that two sets are equal if their elements are equal. We define:

$$\begin{aligned}\mathsf{eq}\,(\mathsf{T}\,X\,f)\,(\mathsf{T}\,Y\,g) &\triangleq (\forall x : X.\,\exists y : Y.\,\mathsf{eq}\,(fx)\,(gy)) \text{ and } (\forall y : Y.\,\exists x : X.\,\mathsf{eq}\,(fx)\,(gy))\\ \mathsf{in}\,a\,(\mathsf{T}\,Y\,g) &\triangleq \exists y : Y.\,\mathsf{eq}\,a\,(gy)\end{aligned}$$

Aczel trees equipped with the above notions of equality and membership form give rise to a constructive set theory. For example, we can define the usual ZF set operations by:

$$\begin{aligned}\emptyset &\triangleq \mathsf{T}\ \bot\ \mathsf{abort}\\ \{a,b\} &\triangleq \mathsf{T}\ \mathbb{B}\ (b \mapsto \mathsf{if}\,b\,\mathsf{then}\,a\,\mathsf{else}\,b)\\ \mathsf{map}\ f\ (\mathsf{T}\,X\,g) &\triangleq \mathsf{T}\ X\ (x \mapsto f(gx))\\ \mathsf{filter}\ P\ (\mathsf{T}\,X\,f) &\triangleq \mathsf{T}\ (\Sigma x : X.\,Px)\ ((x,\_) \mapsto fx)\\ \bigcup(\mathsf{T}\,X\,f) &\triangleq \mathsf{T}\ (\Sigma x : X.\,\pi_1(fx))((x,a) \mapsto (\pi_2(fx))a)\\ \mathcal{P}(\mathsf{T}\,X\,f) &\triangleq \mathsf{T}\ (X \to \mathit{Prop})(P \mapsto \mathsf{T}(\Sigma x : X.\,Px)((x,\_) \mapsto fx))\end{aligned}$$

Non-Constructive Set Theory If we were to use the constructive version of set theory defined above, then, in practice, we would need to ensure that every predicate, every function, and every definition is compatible with the custom notion of equality eq defined above. Instead, we use classical axioms to derive a set theory where the notion of equality coincides with syntactic equality. Using the *axiom of choice, propositional extensionality, and functional extensionality*, we obtain a normalizer $\eta : \mathsf{Acz} \to \mathsf{Acz}$ that returns the canonical representative for a given Aczel tree, meaning:

$$\forall a.\ \mathsf{eq}\,a\,(\eta a) \qquad \text{(NORMAL-EQUAL)}$$

$$\forall a, b.\ \mathsf{eq}\,a\,b \Rightarrow \eta a = \eta b \qquad \text{(NORMAL-EXT)}$$

We use the normalizer η to quotient the type of Aczel trees Acz with the equivalence relation eq and define:

$$\mathsf{Set} \triangleq \Sigma a : \mathsf{Acz}.\ a = \eta a$$

All of the constructive set operations lift to Set, including membership which we denote with $\in$ for values of type Set. Additionally, we obtain the following extensionality principle[3]:

Lemma 1. *If $A \subseteq B$ and $B \subseteq A$, then $A = B$ where $A \subseteq B \triangleq \forall x \in A. x \in B$.*

In particular, this means that Set fulfills the usual ZF axioms and we can operate on it as in set theory.

Ordinals Using standard constructions from set-theory (see *e.g.,* Smullyan and Fitting [2010, Chapters 5 and 6]), formalized by Kirst [2014], we define ordinals in set-theory using the von-Neumann approach and obtain a predicate $\mathsf{isOrdinal} : \mathsf{Set} \to \mathit{Prop}$. That is, each ordinal α is the set of all smaller ordinals, meaning:

$$\alpha = \{\beta \mid \beta \prec \alpha\}$$

Definition 4. *We define $\mathcal{I}_{\mathsf{Ord}} \triangleq (\mathsf{Ord}, \prec_{\mathsf{Ord}}, \mathsf{S}_{\mathsf{Ord}}, 0_{\mathsf{Ord}})$ where:*

$$\mathsf{Ord} \triangleq \Sigma(s : \mathsf{Set}).\mathit{isOrdinal}(s) \qquad \alpha \prec_{\mathsf{Ord}} \beta \triangleq \alpha \in \beta \qquad \mathsf{S}_{\mathsf{Ord}}\, \alpha \triangleq \alpha \cup \{\alpha\} \qquad 0_{\mathsf{Ord}} \triangleq \emptyset$$

Further, we define a limit operation $\lim_{x:X} fx \triangleq \bigcup \{fx \mid x : X\}$ which yields the supremum of all the ordinals fx for $x : X$. For example, we obtain ω, the first infinite ordinal, as $\omega \triangleq \lim_{n:\mathbb{N}} \mathsf{S}^n\, 0$.

Below, we will use this crucial operation to obtain the small existential property for ordinals.

2.3 Existential Properties

From the perspective of the logic, the interesting differences between the step-indexing types defined above all arise in the context of existential quantification. In the remainder of this section, we focus on their differences with respect to existential quantification on a high level, before we descend into the details of the model of Iris propositions (including ordered families of equivalences, resource algebras, and uniform predicates, …) in the next sections. To this end, we assume some step-index type $(\mathcal{I}, \prec, \mathsf{S}, 0)$ and focus on *step-indexed propositions* $\mathsf{sProp}_{\mathcal{I}}$ instead of actual propositions in Iris[4]. Step-indexed propositions $\mathsf{sProp}_{\mathcal{I}}$ are predicates of type $\mathcal{I} \to \mathit{Prop}$ which are down-closed, meaning for a step-indexed proposition $P : \mathsf{sProp}_{\mathcal{I}}$, we have $\forall \alpha, \beta.\, \alpha \prec \beta \Rightarrow P\beta \Rightarrow P\alpha$.

To adequately discuss the differences between the step-index types, we equip step-indexed propositions with a notion of true, false, a later operation, and existential quantification:

$$\top \triangleq \alpha \mapsto \mathsf{True}$$
$$\bot \triangleq \alpha \mapsto \mathsf{False}$$
$$\triangleright P \triangleq \alpha \mapsto \forall \beta \prec \alpha.\, P\,\beta$$
$$\exists x : X.\, \Phi\, x \triangleq \alpha \mapsto \exists x : X.\, (\Phi\, x)\, \alpha$$

We say a proposition $P : \mathsf{sProp}$ *entails* a proposition $Q : \mathsf{sProp}$, written $P \vdash Q$, if $P\,\alpha$ implies $Q\,\alpha$ for all α. We say a proposition $P : \mathsf{sProp}$ is *valid*, written $\vdash P$, if $\forall \alpha.\, P\alpha$.

We highlight the subtle interplay of existential quantification and step-indexing by considering an example, trying to prove the following:

$$\vdash \triangleright \bot \qquad \vdash \triangleright^n \bot \qquad \vdash \exists n : \mathbb{N}.\, \triangleright^n \bot$$

[3] Using proof irrelevance, a corollary of propositional extensionality.

[4] This is done for presentation purposes as the handling of resources is orthogonal to the step-indexing related issues considered here. In the Coq mechanization, there is no concept corresponding to sProp and we use Iris propositions iProp instead.

For natural numbers, we know that it is impossible to prove $\vdash \triangleright \bot$ and $\vdash \triangleright^n \bot$. The reason is that in the former case, we can pick $\alpha = 1$ to obtain a contradiction and in the later case, we can pick $\alpha = n$. The situation changes drastically if we existentially quantify *within the logic* over the number of laters. More precisely, the statement $\vdash \exists n : \mathbb{N}.\ \triangleright^n \bot$ is, in fact, provable. Unfolding the definitions, the statement reads $\forall m.\ \exists n.\ (\triangleright^n \bot)\, m$. For step-index m, we can pick the witness $n \triangleq m + 1$. The step-index m reaches 0 after m later operations and we trivially have $(\triangleright \bot)\, 0$.

For a transfinite step-indexing type, both $\vdash \triangleright \bot$ and $\vdash \triangleright^n \bot$, but also $\vdash \exists n : \mathbb{N}.\ \triangleright^n \bot$ are impossible to prove. To see why $\vdash \exists n : \mathbb{N}.\ \triangleright^n \bot$ is impossible to prove, we consider what happens at step-index ω. Unfolding the definitions, at step-index ω, some witness n must have been chosen such that $(\triangleright^n \bot)\, \omega$. Thus, we have $\forall m.(\triangleright^n \bot)\, m$ by downward-closure which we know is impossible to prove.

Abstractly, the interaction between step-indexing and existential quantification boils down to the following question: "When can a quantifier over step-indices be commuted with an existential quantifier?". More formally, for which choices of X and $\Phi : X \to \mathsf{sProp}$ do the following rules hold?

$$\frac{\forall \alpha.\ \exists x : X.\ (\Phi x)\alpha}{\exists x : X.\ \forall \alpha.\ (\Phi x)\alpha} \qquad \frac{\forall \beta \prec \alpha.\ \exists x : X.\ (\Phi x)\beta}{\exists x : X.\ \forall \beta \prec \alpha.\ (\Phi x)\beta}$$

We refer to the former as the *existential property* and to the latter as the *bounded existential property*.

Bounded Existential Property Of the step-indexing types introduced above, natural numbers are the only type which satisfies the bounded existential property for all inhabited types X and all choices of Φ. The reason is that every index except for zero has a largest predecessor. To find the right witness x, we can just use the witness of the largest predecessor which is sufficient for smaller step-indices by down-closure.

Transfinite step-index types cannot enjoy the bounded existential property. For transfinite step-index types, we have already seen in the example above that the proposition $\exists x : X.\ \forall \beta \prec \omega.\ (\Phi x)\beta$ is false for $X \triangleq \mathbb{N}$ and $\Phi n \triangleq \triangleright^n \bot$. However, $\forall \beta \prec \omega.\ \exists x : X.\ (\Phi x)\beta$ is true for $X \triangleq \mathbb{N}$ and $\Phi n \triangleq \triangleright^n \bot$ with $x \triangleq \beta + 1$[5] as we also argued above.

The bounded existential property is equivalent to the following entailment, unfolding the definitions of the operations on sProp:

$$\frac{X \text{ is a non-empty type}}{\triangleright(\exists x : X.\ \Phi x) \vdash (\exists x : X.\ \triangleright \Phi x)}$$

Consequently, any transfinite step-indexing type cannot prove soundness of the commuting rule of later with existential quantification.

Existential Property By definition, the existential property is equivalent to the following (meta-level) rule:

$$\frac{\vdash \exists x : X.\ \Phi x}{\exists x : X.\ \vdash \Phi x}$$

The example above shows that natural numbers do not enjoy the general existential property for all types X and predicates Φ. More precisely, we know that $\vdash \exists n : \mathbb{N}. \triangleright^n \bot$ is provable. However, $\exists n : \mathbb{N}. \vdash \triangleright^n \bot$ cannot be true since for each n, we know that $\vdash \triangleright^n \bot$ is false.

This fact suggests a certain relation between the size of the step-index type and the size of the types X for which we can validate the existential property. In principle, one may therefore expect that any step-index type $\mathcal{I}$ cannot validate the existential property for types X which are at least as large as $\mathcal{I}$.

[5] Technically, the embedding of β into the natural numbers.

However, in practice in proofs many existential quantifiers are not instantiated based on the current step-index. As we shall see in subsequent sections, it can be really useful to have existential properties in these cases. In fact, we can recover existential properties for specific choices of X and Φ.

Finite Existential Properties For finite existential quantifiers, we can recover the existential properties classically. That is, we can show (under the assumption of the law of excluded middle):

$$\triangleright P \vee Q \vdash (\triangleright P) \vee (\triangleright Q) \qquad\qquad \frac{\vdash P \vee Q}{\vdash P \text{ or } \vdash Q}$$

where $P \vee Q \triangleq \exists b : \mathbb{B}.\, \mathsf{if}\, b\, \mathsf{then}\, P\, \mathsf{else}\, Q$. For finite sets other than $\mathbb{B}$, we can use repeated application of the rules for disjunction to obtain the existential properties. We showcase the proof of the latter property:

Lemma 2. *If* $\vdash P \vee Q$*, then* $\vdash P$ *or* $\vdash Q$*.*

Proof. Let $\vdash P \vee Q$. If $\vdash P$, then the claim is trivial. Let $\neg \vdash P$. Then there is some step-index α such that $\neg P\alpha$. By down-closure, we know $\neg P\beta$ for all β with $\alpha \preceq \beta$. Since $\vdash P \vee Q$, we know $Q\beta$ for all β with $\alpha \preceq \beta$. By down-closure, we also know $Q\beta$ for all $\beta \prec \alpha$. Thus, we have shown $\vdash Q$. □

Small Existential Property As was suggested above, in general it does not seem possible to prove the general existential property:

$$\frac{\vdash \exists x : X.\, \Phi x}{\exists x : X.\ \vdash \Phi x}$$

However, as we shall see below, for the non-constructive ordinals, we can show the property for all types X which are significantly smaller in cardinality than the ordinal type Ord. To explain what significantly smaller means and how we can obtain such a proof, we take a closer look at the universe mechanism underlying our meta logic.

To this end, assume the meta theory does not contain a hierarchy of universes but only a single one Type and recall the definition of Aczel trees:

$$a, b : \mathsf{Acz} ::= \mathsf{T}(X : \mathit{Type})(f : X \to \mathsf{Acz})$$

What happens if we construct the "tree of all trees" (in analogy to the set of all sets)? That is, we define $a_{\mathsf{Acz}} \triangleq \mathsf{T}\ \mathsf{Acz}\ (a \mapsto a)$. In this case, we obtain $\forall a.\neg \mathsf{in}\, a\, a$ by induction on a, but also $\mathsf{in}\, a_{\mathsf{Acz}} a_{\mathsf{Acz}}$, an inconsistency.

We can avoid this inconsistency by using a hierarchy of universes $\mathit{Type}_0 : \mathit{Type}_1 : \mathit{Type}_2 \ldots$ The hierarchy allows us to place the type of Aczel trees Acz one universe above the type X. That is, if $X : \mathit{Type}_i$, then $\mathsf{T}\, X f : \mathsf{Acz}$ where $\mathsf{Acz} : \mathit{Type}_{i+1}$. By extension, the types Set and Ord are in universe i if they are constructed using Aczel trees in universe i.

The *small existential property* gives us a soundness result for quantifiers $X : \mathit{Type}_i$ where the step-index type $\mathcal{I}$ is $\mathsf{Ord} : \mathit{Type}_{i+1}$, written Ord_{i+1}.

Lemma 3. *Let* $X : \mathit{Type}_i$ *and* $\phi : X \to \mathsf{Ord}_{i+1} \to \mathit{Prop}$ *such that* ϕx *is up-closed for all* x*, meaning* $\forall x \alpha \beta.\, \alpha \preceq \beta \Rightarrow \phi\, x\, \alpha \Rightarrow \phi\, x\, \beta$*. If* $\forall x : X.\, \exists \alpha : \mathsf{Ord}_{i+1}.\, (\phi x)\, \alpha$*, then* $\exists \alpha : \mathsf{Ord}_{i+1}.\, \forall x : X.\, (\phi x)\, \alpha$*.*

Proof. Using the axiom of choice, there is a function $f : X \to \mathsf{Ord}_{i+1}$ such that $\forall x.\, (\phi x)\, (f x)$. We define $\alpha : \mathsf{Ord}_{i+1} \triangleq \lim_{x:X} f\, x$. It remains to show $\forall x : X.\, (\phi x)\, \alpha$ which follows by up-closure, given that $f x \preceq \alpha : \mathsf{Ord}_{i+1} \triangleq \lim_{x:X} f\, x$ for all $x : X$. □

Lemma 4 (Small Existential Property)**.** *Let* $X : \mathit{Type}_i$ *and the step-index type be* Ord_{i+1}.

$$\frac{\vdash \exists x : X.\, \Phi x}{\exists x : X.\ \vdash \Phi x}$$

Proof. Unfolding the definitions, assume $\forall \alpha : \mathsf{Ord}_{i+1}.\ \exists x : X.\ (\Phi x)\ \alpha$. By way of contradiction, assume $\neg \exists x : X.\ \vdash \Phi x$. That is (classically), $\forall x : X.\ \exists \alpha : \mathsf{Ord}_{i+1}.\ \neg(\Phi x)\alpha$. The predicate $x \mapsto \alpha \mapsto \neg(\Phi x)\alpha$ is up-closed, since Φx is down-closed for all x. By Lemma 3, we have $\exists \alpha : \mathsf{Ord}_{i+1}.\ \forall x : X.\ \neg(\Phi x)\ \alpha$, a contradiction to $\forall \alpha : \mathsf{Ord}_{i+1}.\ \exists x : X.\ (\Phi x)\ \alpha$. □

Existential Properties Overview We summarize the results of this section in the following table:

Step-Index Type	Finite Bounded	Bounded	Finite	Small
Natural Numbers	yes	yes	classically	no
Constructive Transfinite (*e.g.,* ω^n)	classically	no	classically	no[6]
Non-Constructive Transfinite	yes	no	yes	yes

[6] not mechanized

3 Model

3.1 Ordered Families of Equivalences

The model of finite Iris lives in the category of *Ordered Families of Equivalences* (OFEs). For transfinite Iris, we generalize OFEs to arbitrary step-indexing types. To this end let $\mathcal{I}$ be an arbitrary step-index type.

Definition 5. *An* ordered family of equivalences *(OFE) is a tuple* $(T/_{\equiv}, (\overset{\alpha}{=} : T \to T \to \mathit{Prop})_{\alpha:\mathcal{I}})$ *satisfying*

$$\forall \alpha.\ (\overset{\alpha}{=}) \text{ is an equivalence relation} \tag{OFE-EQUIV}$$

$$\forall \alpha, \beta.\ \beta \prec \alpha \Rightarrow (\overset{\alpha}{=}) \subseteq (\overset{\beta}{=}) \tag{OFE-MONO}$$

where $x \equiv y \triangleq \forall \alpha. x \overset{\alpha}{=} y$.

The idea of this definition is that larger step-indices α may be understood as more "time" to computationally distinguish two objects. In other words, as α increases, $\overset{\alpha}{=}$ becomes more and more refined (OFE-MONO)—and in the limit, it agrees with plain equality, due to the required quotienting[7].

Definition 6. *An value* $x : T$ *of an OFE is called* discrete *if*

$$\forall y : T.\ x \overset{0}{=} y \Rightarrow x = y$$

An OFE A *is called* discrete *if all its elements are discrete. For a type* X*, we write* ΔX *for the discrete OFE with* $x \overset{\alpha}{=} x' \triangleq x = x'$.

Definition 7. *A function* $f : T \to U$ *between two OFEs is* non-expansive *(written* $f : T \overset{ne}{\to} U$*) if*

$$\forall \alpha.\ \forall x, y : T.\ x \overset{\alpha}{=} y \Rightarrow f(x) \overset{\alpha}{=} f(y)$$

It is contractive *if*

$$\forall \alpha.\ \forall x, y : T.\ (\forall \beta \prec \alpha.\ x \overset{\beta}{=} y) \Rightarrow f(x) \overset{\alpha}{=} f(y)$$

Intuitively, applying a non-expansive function to some data will not suddenly introduce differences between seemingly equal data. A contractive function will make seemingly equal data even more indistinguishable.

Definition 8. *The category* **OFE** *consists of OFEs as objects, and non-expansive functions as arrows.*

Note that **OFE** is bicartesian closed, *i.e.,* it has all sums, products and exponentials as well as an initial and a terminal object. In particular:

Definition 9. *Given two OFEs* T *and* U*, the non-expansive function space* $T \overset{ne}{\to} U$ *is itself an OFE with*

$$f \overset{\alpha}{=} g \triangleq \forall x : T.\ f(x) \overset{\alpha}{=} g(x)$$

Note that with this definition of step-indexed equality for function-space OFEs, functions $f : T \overset{\text{ne}}{\to} U$ *are* extensional, even though we do not assume functional extensionality at the meta-level[8].

[7] Formally, according to Section 1.1, this just means that we define a setoid structure with $\equiv$ on top of the arbitary type T and overload the equality symbol.

[8] Formally, this works due to the implicit "quotienting" (*i.e.,* the setoid structure on top of OFEs).

Definition 10. *A (bi)functor $F : \mathbf{OFE} \to \mathbf{OFE}$ is called* locally non-expansive *if its action F_1 on arrows is itself a non-expansive map. Similarly, F is called* locally contractive *if F_1 is a contractive map.*

The function space $(-) \xrightarrow{\text{ne}} (-)$ is a locally non-expansive bifunctor. Note that the composition of non-expansive (bi)functors is non-expansive, and the composition of a non-expansive and a contractive (bi)functor is contractive.

One very important OFE is the OFE of *step-indexed propositions* sProp, which we have already encountered in Section 2.3: For every step-index, such a proposition either holds or does not hold. Moreover, if a proposition holds for some n, written as $n \in P$, it also has to hold for all smaller step-indices.

$$\mathsf{sProp} \triangleq \Sigma P : \mathbb{N} \to \mathit{Prop}. \forall \alpha\beta.\ \alpha \preceq \beta \Rightarrow P\,\beta \Rightarrow P\,\alpha$$
$$P \overset{\alpha}{=} Q \triangleq \forall \beta \preceq \alpha.\ P\,\beta \Leftrightarrow Q\,\beta$$
$$P \overset{\alpha}{\subseteq} Q \triangleq \forall \beta \preceq \alpha.\ P\,\beta \Rightarrow Q\,\beta$$

3.2 Complete Ordered Families of Equivalences

COFEs are *complete OFEs*, which means that we can take limits of *chains*.

Definition 11 (Chain)**.** *Given some OFE T, a* chain *is a function $c : \mathcal{I} \to T$ such that $\forall \alpha, \beta.\ \alpha \preceq \beta \Rightarrow c(\beta) \overset{\alpha}{=} c(\alpha)$.*

In the transfinite setting, we not only need to be able to take limits of chains but also of *bounded chains*, that is chains which are bounded by an index α.

Definition 12 (Bounded Chain)**.** *Given some OFE T, an* α-bounded chain *is a function $c : \Pi\beta.\ \beta \prec \alpha \to T$ such that $\forall \beta \prec \alpha, \gamma \prec \alpha.\ \beta \preceq \gamma \Rightarrow c(\gamma) \overset{\beta}{=} c(\beta)$.*

Definition 13. *A* complete ordered family of equivalences *(COFE) is a tuple $(T : \mathbf{OFE}, \lim : \mathrm{chain}(T) \to T, \mathrm{blim} : \Pi\alpha \succ 0.\ \mathrm{bchain}_\alpha(T) \to T)$ satisfying*

$$\forall \alpha, (c : \mathrm{chain}(X)).\ \lim(c) \overset{\alpha}{=} c(\alpha) \qquad \text{(COFE-COMPL)}$$
$$\forall \alpha \succ 0, \beta \prec \alpha, (c : \mathrm{bchain}_\alpha(X)).\ \mathrm{blim}(c) \overset{\beta}{=} c(\beta) \qquad \text{(COFE-BCOMPL)}$$
$$\forall \alpha \succ 0, \beta.\ \forall (c, d : \mathrm{bchain}_\alpha(X)).\ (\forall \gamma \prec \alpha. c(\gamma) \overset{\beta}{=} d(\gamma)) \Rightarrow \mathrm{blim}(c) \overset{\beta}{=} \mathrm{blim}(d) \qquad \text{(COFE-BCOMPL-NE)}$$

Definition 14. *The category* **COFE** *consists of COFEs as objects, and non-expansive functions as arrows.*

The function space $T \xrightarrow{\text{ne}} U$ is a COFE if U is a COFE (*i.e.,* the domain T can actually be just an OFE). sProp as defined above is complete, *i.e.,* it is a COFE.

Completeness is necessary to take fixed-points.

Theorem 1 (Banach's fixed-point)**.** *Given an inhabited COFE T and a contractive function $f : T \to T$, there exists a unique fixed-point $\mathit{fix}_T f$ such that $f(\mathit{fix}_T f) = \mathit{fix}_T f$. Moreover, this theorem also holds if f is just non-expansive and f^k is contractive for an arbitrary k.*

3.3 Resource Algebras

Definition 15. *A* resource algebra *(RA) is a tuple* $(M, \overline{\mathcal{V}} : M \to \mathit{Prop}, |-| : M \to M^?, (\cdot) : M \times M \to M)$ *satisfying:*

$$\forall a, b, c.\ (a \cdot b) \cdot c = a \cdot (b \cdot c) \qquad \text{(RA-ASSOC)}$$

$$\forall a, b.\ a \cdot b = b \cdot a \qquad \text{(RA-COMM)}$$

$$\forall a.\ |a| : M \Rightarrow |a| \cdot a = a \qquad \text{(RA-CORE-ID)}$$

$$\forall a.\ |a| : M \Rightarrow ||a|| = |a| \qquad \text{(RA-CORE-IDEM)}$$

$$\forall a, b.\ |a| : M \land a \preccurlyeq b \Rightarrow |b| : M \land |a| \preccurlyeq |b| \qquad \text{(RA-CORE-MONO)}$$

$$\forall a, b.\ \overline{\mathcal{V}}(a \cdot b) \Rightarrow \overline{\mathcal{V}}(a) \qquad \text{(RA-VALID-OP)}$$

where $a^? \cdot \bot \triangleq \bot \cdot a^? \triangleq a^?$

$$a \preccurlyeq b \triangleq \exists c : M.\ b = a \cdot c \qquad \text{(RA-INCL)}$$

RAs are closely related to *Partial Commutative Monoids* (PCMs), with two key differences:

1. The composition operation on RAs is total (as opposed to the partial composition operation of a PCM), but there is a specific subset of *valid* elements that is compatible with the composition operation (RA-VALID-OP). These valid elements are identified by the *validity predicate* $\overline{\mathcal{V}}$.

   This take on partiality is necessary when defining the structure of *higher-order* ghost state, *cameras*, in the next subsection.

2. Instead of a single unit that is an identity to every element, we allow for an arbitrary number of units, via a function $|-|$ assigning to an element a its *(duplicable) core* $|a|$, as demanded by RA-CORE-ID. We further demand that $|-|$ is idempotent (RA-CORE-IDEM) and monotone (RA-CORE-MONO) with respect to the *extension order*, defined similarly to that for PCMs (RA-INCL).

   Notice that the codomain of the core is the option type $M^?$, adding a dummy element $\bot$ to M. Thus, the core can be *partial*: not all elements need to have a unit. Partial cores help us to build interesting composite RAs from smaller primitives. We use the metavariable $a^?$ to indicate elements of $M^?$. We also lift the composition $(\cdot)$ to $M^?$. In a slight abuse of notation, we write $a^? : M$ for the assertion that there exists $a' : M$ with

   Notice also that the core of an RA is a strict generalization of the unit that any PCM must provide, since $|-|$ can always be picked as a constant function.

Definition 16. *It is possible to do a* frame-preserving update *from* $a : M$ *to elements satisfying* $P : M \to \mathit{Prop}$*, written* $a \rightsquigarrow P$*, if*

$$\forall a_{\mathrm{f}}^? : M^?.\ \overline{\mathcal{V}}(a \cdot a_{\mathrm{f}}^?) \Rightarrow \exists b : M.\ P\ b \land \overline{\mathcal{V}}(b \cdot a_{\mathrm{f}}^?)$$

We further define $a \rightsquigarrow b \triangleq \forall a_{\mathrm{f}}^? : M^?.\ \overline{\mathcal{V}}(a \cdot a_{\mathrm{f}}^?) \Rightarrow \overline{\mathcal{V}}(b \cdot a_{\mathrm{f}}^?)$.

The proposition $a \rightsquigarrow P$ says that every element $a_{\mathrm{f}}^?$ compatible with a (we also call such elements *frames*), must also be compatible with some b satisfying $P\ b$. Notice that $a_{\mathrm{f}}^?$ could be $\bot$, so the frame-preserving update can also be applied to elements that have *no* frame. Intuitively, this means that whatever assumptions the rest of the program is making about the state of γ, if these assumptions are compatible with a, then updating to b will not invalidate any of these assumptions. Since Iris ensures that the global ghost state is valid, this means that we can soundly update the ghost state from a to a non-deterministically picked b with $P\ b$.

3.4 Cameras

Definition 17. *A* camera *is a tuple* $(M : \mathbf{OFE}, \mathcal{V} : M \xrightarrow{ne} \mathsf{sProp}, |-| : M \xrightarrow{ne} M^?, (\cdot) : M \times M \xrightarrow{ne} M)$ *satisfying:*

$$\forall a, b, c.\ (a \cdot b) \cdot c = a \cdot (b \cdot c) \qquad \text{(CAMERA-ASSOC)}$$

$$\forall a, b.\ a \cdot b = b \cdot a \qquad \text{(CAMERA-COMM)}$$

$$\forall a.\ |a| : M \Rightarrow |a| \cdot a = a \qquad \text{(CAMERA-CORE-ID)}$$

$$\forall a.\ |a| : M \Rightarrow ||a|| = |a| \qquad \text{(CAMERA-CORE-IDEM)}$$

$$\forall a, b.\ |a| : M \land a \preccurlyeq b \Rightarrow |b| : M \land |a| \preccurlyeq |b| \qquad \text{(CAMERA-CORE-MONO)}$$

$$\forall a, b.\ \mathcal{V}(a \cdot b) \subseteq \mathcal{V}(a) \qquad \text{(CAMERA-VALID-OP)}$$

$$\forall \alpha, a, b_1, b_2.\ \alpha \in \mathcal{V}(a) \land a \overset{\alpha}{=} b_1 \cdot b_2 \Rightarrow \Sigma c_1, c_2.\ a = c_1 \cdot c_2 \land c_1 \overset{\alpha}{=} b_1 \land c_2 \overset{\alpha}{=} b_2 \qquad \text{(CAMERA-EXTEND)}$$

where

$$a \preccurlyeq b \triangleq \Sigma c.\ b = a \cdot c \qquad \text{(CAMERA-INCL)}$$

$$a \overset{\alpha}{\preccurlyeq} b \triangleq \Sigma c.\ b \overset{\alpha}{=} a \cdot c \qquad \text{(CAMERA-INCLN)}$$

This is a natural generalization of RAs over OFEs[9]. All operations have to be non-expansive, and the validity predicate $\mathcal{V}$ can now also depend on the step-index. We define the plain $\overline{\mathcal{V}}$ as the "limit" of the step-indexed approximation:

$$\overline{\mathcal{V}}(a) \triangleq \forall \alpha.\ \alpha \in \mathcal{V}(a)$$

The extension axiom (CAMERA-EXTEND). Notice that the existential quantification in this axiom is *constructive, i.e.,* it is a sigma type in Coq. The purpose of this axiom is to compute a_1, a_2 completing the following square:

$$\begin{array}{ccc} a & \overset{\alpha}{=} & b \\ \| & & \| \\ a_1 \cdot a_2 & \overset{\alpha}{=} & b_1 \cdot b_2 \end{array}$$

where the α-equivalence at the bottom is meant to apply to the pairs of elements, *i.e.,* we demand $a_1 \overset{\alpha}{=} b_1$ and $a_2 \overset{\alpha}{=} b_2$. In other words, extension carries the decomposition of b into b_1 and b_2 over the α-equivalence of a and b, and yields a corresponding decomposition of a into a_1 and a_2.

With *finite step-indices*, this operation is needed to prove that $\triangleright$ commutes with separating conjunction:

$$\triangleright(P * Q) \Leftrightarrow \triangleright P * \triangleright Q$$

Transfinite indices do not enjoy this commuting property (see Section 4).

[9] The reader may wonder why on earth we call them "cameras". The reason, which may not be entirely convincing, is that "camera" was originally just used as a comfortable pronunciation of "CMRA", the name used in earlier Iris papers. CMRA was originally supposed to be an acronym for "complete metric resource algebras" (or something like that), but we were never very satisfied with it and thus ended up never spelling it out. To make matters worse, the "complete" part of CMRA is now downright misleading, for whereas previously the carrier of a CMRA was required to be a COFE (complete OFE), we have relaxed that restriction and permit it to be an (incomplete) OFE. For these reasons, we have decided to stick with the name "camera", for purposes of continuity, but to drop any pretense that it stands for something.

Definition 18. *An element ε of a camera M is called the* unit *of M if it satisfies the following conditions:*

1. *ε is valid:*
   $\forall \alpha.\, \alpha \in \mathcal{V}(\varepsilon)$
2. *ε is a left-identity of the operation:*
   $\forall a \in M.\, \varepsilon \cdot a = a$
3. *ε is its own core:*
   $|\varepsilon| = \varepsilon$

Lemma 5. *If M has a unit ε, then the core $|-|$ is total,* i.e., $\forall a.\, |a| \in M$.

Definition 19. *It is possible to do a* frame-preserving update *from $a \in M$ to elements satisfying $P : M \to$ Prop, written $a \rightsquigarrow P$, if*

$$\forall \alpha, a_{\mathrm{f}}^{?}.\, \alpha \in \mathcal{V}(a \cdot a_{\mathrm{f}}^{?}) \Rightarrow \exists b : M.\, P\, b \wedge \alpha \in \mathcal{V}(b \cdot a_{\mathrm{f}}^{?})$$

We further define $a \rightsquigarrow b \triangleq \forall \alpha, a_{\mathrm{f}}^{?}.\, \alpha \in \mathcal{V}(a \cdot a_{\mathrm{f}}^{?}) \Rightarrow \alpha \in \mathcal{V}(b \cdot a_{\mathrm{f}}^{?})$.

Note that for RAs, this and the RA-based definition of a frame-preserving update coincide.

Definition 20. *A camera M is* discrete *if it satisfies the following conditions:*

1. *M is a discrete OFE*
2. *$\mathcal{V}$ ignores the step-index:*
   $\forall a \in M.\, 0 \in \mathcal{V}(a) \Rightarrow \forall \alpha.\, \alpha \in \mathcal{V}(a)$

Note that every RA is a discrete camera, by picking the discrete OFE for the equivalence relation. Furthermore, discrete cameras can be turned into RAs by ignoring their OFE structure, as well as the step-index of $\mathcal{V}$.

Definition 21 (Camera homomorphism)**.** *A non-expansive function $f : M_1 \xrightarrow{ne} M_2$ between two cameras is* a camera homomorphism *if it satisfies the following conditions:*

1. *f commutes with composition:*
   $\forall a_1 : M_1, a_2 : M_1.\, f(a_1) \cdot f(a_2) = f(a_1 \cdot a_2)$
2. *f commutes with the core:*
   $\forall a : M_1.\, |f(a)| = f(|a|)$
3. *f preserves validity:*
   $\forall \alpha.\, \forall a : M_1.\, \alpha \in \mathcal{V}(a) \Rightarrow \alpha \in \mathcal{V}(f(a))$

Definition 22. *The category* **Camera** *consists of cameras as objects, and camera homomorphisms as arrows.*

Note that every object/arrow in **Camera** is also an object/arrow of **OFE**. The notion of a locally non-expansive (or contractive) bifunctor naturally generalizes to bifunctors between these categories.

3.5 Constructions

3.5.1 Trivial Pointwise Lifting

The (C)OFE structure on many types can be easily obtained by pointwise lifting of the structure of the components. This is what we do for option $T^{?}$, product $(M_i)_{i \in I}$ (with I some finite index set), sum $T + T'$, and finite partial functions $K \xrightarrow{\text{fin}} M$ over some countable domain K.

3.5.2 Next (Type-Level Later)

Given a OFE T, we define $\blacktriangleright T$ as follows (using a datatype-like notation to define the type):

$$\blacktriangleright T \triangleq \mathsf{next}(x : T)$$
$$\mathsf{next}(x) \overset{\alpha}{=} \mathsf{next}(y) \triangleq \forall \beta \prec \alpha. x \overset{\beta}{=} y$$

$\blacktriangleright(-)$ is a locally *contractive* functor from **OFE** to **OFE**.

3.5.3 Product Camera

Given a family $(M_i)_{i \in I}$ of cameras (I finite), we construct a camera for the product $\prod_{i \in I} M_i$ by lifting everything pointwise.

Frame-preserving updates on the M_i lift to the product:

$$\textsc{prod-update} \quad \frac{a \rightsquigarrow_{M_i} P}{f\,[i \leftarrow a] \rightsquigarrow \lambda x. x = f\,[i \leftarrow b] \wedge P\,b}$$

3.5.4 Sum Camera

The *sum camera* $M_1 +_\lightning M_2$ for any cameras M_1 and M_2 is defined as:

$$M_1 +_\lightning M_2 \triangleq \mathsf{inl}(a_1 : M_1) \mid \mathsf{inr}(a_2 : M_2) \mid \lightning$$
$$\mathcal{V}(\lightning) \triangleq \text{False}$$
$$\mathcal{V}(\mathsf{inl}(a)) \triangleq \mathcal{V}_1(a)$$
$$\mathsf{inl}(a_1) \cdot \mathsf{inl}(b_1) \triangleq \mathsf{inl}(a_1 \cdot b_1)$$
$$|\mathsf{inl}(a_1)| \triangleq \begin{cases} \bot & \text{if } |a_1| = \bot \\ \mathsf{inl}(|a_1|) & \text{otherwise} \end{cases}$$

Above, $\mathcal{V}_1$ refers to the validity of M_1. The validity, composition and core for inr are defined symmetrically. The remaining cases of the composition and core are all $\lightning$.

Notice that we added the artificial "invalid" (or "undefined") element $\lightning$ to this camera just in order to make certain compositions of elements (in this case, inl and inr) invalid.

The step-indexed equivalence is inductively defined as follows:

$$\frac{x \overset{\alpha}{=} y}{\mathsf{inl}(x) \overset{\alpha}{=} \mathsf{inl}(y)} \qquad \frac{x \overset{\alpha}{=} y}{\mathsf{inr}(x) \overset{\alpha}{=} \mathsf{inr}(y)} \qquad \lightning \overset{\alpha}{=} \lightning$$

We obtain the following frame-preserving updates, as well as their symmetric counterparts:

$$\textsc{sum-update} \quad \frac{a \rightsquigarrow_{M_1} P}{\mathsf{inl}(a) \rightsquigarrow \lambda x. x = \mathsf{inl}(b) \wedge P\,b} \qquad \textsc{sum-swap} \quad \frac{\forall a_\mathrm{f} \in M, \alpha.\, \alpha \notin \mathcal{V}(a \cdot a_\mathrm{f}) \qquad \overline{\mathcal{V}}(b)}{\mathsf{inl}(a) \rightsquigarrow \mathsf{inr}(b)}$$

Crucially, the second rule allows us to *swap* the "side" of the sum that the camera is on if $\mathcal{V}$ has *no possible frame*.

3.5.5 Option Camera

The definition of the camera/RA axioms already lifted the composition operation on M to one on $M^?$. We can easily extend this to a full camera by defining a suitable core, namely

$$\begin{aligned} |\bot| &\triangleq \bot \\ |a^?| &\triangleq |a| && \text{If } a^? = S(a) \end{aligned}$$

Notice that this core is total, as the result always lies in $M^?$ (rather than in $M^{?^?}$).

3.5.6 Finite Partial Functions Camera

Given some countable K and some camera M, the OFE of finite partial functions $K \xrightarrow{\text{fin}} M$ is equipped with a camera structure by lifting everything pointwise.

We obtain the following frame-preserving updates:

FPFN-ALLOC-STRONG
$$\frac{I \text{ infinite} \qquad \overline{\mathcal{V}}(a) \qquad I(\gamma : K)}{\emptyset \rightsquigarrow [\gamma \leftarrow a]}$$

FPFN-ALLOC
$$\frac{\overline{\mathcal{V}}(a) \qquad \gamma : K}{\emptyset \rightsquigarrow [\gamma \leftarrow a]}$$

FPFN-UPDATE
$$\frac{a \rightsquigarrow_M P}{f\,[i \leftarrow a] \rightsquigarrow \lambda x.x = f\,[i \leftarrow b] \wedge P\ b}$$

Above, $\overline{\mathcal{V}}$ refers to the (full) validity of M.

$K \xrightarrow{\text{fin}} (-)$ is a locally non-expansive functor from **Camera** to **Camera**.

3.5.7 Agreement Camera

Given some OFE T, we define the camera $\text{Ag}(T)$ as follows:

$$\begin{aligned} \text{Ag}(T) &\triangleq (\Sigma a : \mathsf{list}(T).\ a \neq [])/\sim \\ \text{where } a \overset{\alpha}{=} b &\triangleq (\forall x \in a.\ \exists y \in b.\ x \overset{\alpha}{=} y) \wedge (\forall y \in b.\ \exists x \in a.\ x \overset{\alpha}{=} y) \\ a \sim b &\triangleq \forall \alpha.\ a \overset{\alpha}{=} b \\ \mathcal{V}(a)(\alpha) &\triangleq \forall x, y \in a.\ x \overset{\alpha}{=} y \\ |a| &\triangleq a \\ a \cdot b &\triangleq a +\!\!+ b \end{aligned}$$

$\text{Ag}(-)$ is a locally non-expansive functor from **OFE** to **Camera**.

We define a non-expansive injection ag into $\text{Ag}(T)$ as follows:

$$\mathsf{ag}(x) \triangleq [x]$$

There are no interesting frame-preserving updates for $\text{Ag}(T)$, but we can show the following:

AG-VAL
$$\overline{\mathcal{V}}(\mathsf{ag}(x))$$

AG-DUP
$$\mathsf{ag}(x) = \mathsf{ag}(x) \cdot \mathsf{ag}(x)$$

AG-AGREE
$$\alpha \in \mathcal{V}(\mathsf{ag}(x) \cdot \mathsf{ag}(y)) \Rightarrow x \overset{\alpha}{=} y$$

3.5.8 Exclusive Camera

Given an OFE T, we define a camera $\mathrm{Ex}(T)$ such that at most one $x \in T$ can be owned:

$$\mathrm{Ex}(T) \triangleq \mathsf{ex}(T) \mid \lightning$$
$$\mathcal{V}(a)(\alpha) \triangleq a \overset{\alpha}{\neq} \lightning$$

All cases of composition go to $\lightning$.

$$|\mathsf{ex}(x)| \triangleq \bot \qquad\qquad |\lightning| \triangleq \lightning$$

Remember that $\bot$ is the "dummy" element in $M^?$ indicating (in this case) that $\mathsf{ex}(x)$ has no core.

The step-indexed equivalence is inductively defined as follows:

$$\frac{x \overset{\alpha}{=} y}{\mathsf{ex}(x) \overset{\alpha}{=} \mathsf{ex}(y)} \qquad\qquad \lightning \overset{\alpha}{=} \lightning$$

$\mathrm{Ex}(-)$ is a locally non-expansive functor from **OFE** to **Camera**.

We obtain the following frame-preserving update:

EX-UPDATE
$$\mathsf{ex}(x) \rightsquigarrow \mathsf{ex}(y)$$

3.5.9 Fractions Camera

We define an RA structure on the rational numbers in $(0, 1]$ as follows:

$$\textsc{Frac} \triangleq \mathsf{frac}(\mathbb{Q} \cap (0, 1]) \mid \lightning$$
$$\overline{\mathcal{V}}(a) \triangleq a \neq \lightning$$
$$\mathsf{frac}(q_1) \cdot \mathsf{frac}(q_2) \triangleq \mathsf{frac}(q_1 + q_2) \quad \text{if } q_1 + q_2 \leq 1$$
$$|\mathsf{frac}(x)| \triangleq \bot$$
$$|\lightning| \triangleq \lightning$$

All remaining cases of composition go to $\lightning$. Frequently, we will write just x instead of $\mathsf{frac}(x)$.

The most important property of this RA is that 1 has no frame. This is useful in combination with SUM-SWAP, and also when used with pairs:

PAIR-FRAC-CHANGE
$$(1, a) \rightsquigarrow (1, b)$$

3.5.10 Monotone List Camera

Given a type A, we define an RA structure on lists of elements of A as follows:

$$\begin{aligned}
\textsc{MList} &\triangleq \mathsf{mlist}(\mathsf{List}A) \mid \lightning \\
\overline{\mathcal{V}}(a) &\triangleq a \neq \lightning \\
\mathsf{mlist}(l_1) \cdot \mathsf{mlist}(l_2) &\triangleq \begin{cases} l1 & \text{if } l_2 \text{ is a prefix of } l_1 \\ l2 & \text{if } l_1 \text{ is a prefix of } l_2 \\ \lightning & \text{otherwise} \end{cases} \\
|\mathsf{mlist}(l)| &\triangleq \mathsf{mlist}(l) \\
|\lightning| &\triangleq \lightning
\end{aligned}$$

Frequently, we will write just l instead of $\mathsf{mlist}(l)$.

3.5.11 Authoritative Camera

Given a camera M, we construct $\textsc{Auth}(M)$ modeling someone owning an *authoritative* element a of M, and others potentially owning fragments $b \preccurlyeq a$ of a. We assume that M has a unit ε, and hence its core is total. (If M is an exclusive monoid, the construction is very similar to a half-ownership monoid with two asymmetric halves.)

$$\begin{aligned}
\textsc{Auth}(M) &\triangleq \textsc{Ex}(M)^? \times M \\
\mathcal{V}((x,b))(\alpha) &\triangleq (x = \bot \land \alpha \in \mathcal{V}(b)) \lor (\exists a.\, x = \mathsf{ex}(a) \land b \preccurlyeq_\alpha a \land \alpha \in \mathcal{V}(a)) \\
(x_1, b_1) \cdot (x_2, b_2) &\triangleq (x_1 \cdot x_2, b_2 \cdot b_2) \\
|(x,b)| &\triangleq (\bot, |b|) \\
(x_1, b_1) \overset{\alpha}{=} (x_2, b_2) &\triangleq x_1 \overset{\alpha}{=} x_2 \land b_1 \overset{\alpha}{=} b_2
\end{aligned}$$

Note that $(\bot, \varepsilon)$ is the unit and asserts no ownership whatsoever, but $(\mathsf{ex}(\varepsilon), \varepsilon)$ asserts that the authoritative element is ε.

Let $a, b \in M$. We write $\bullet\, a$ for full ownership $(\mathsf{ex}(a), \varepsilon)$ and $\circ\, b$ for fragmental ownership $(\bot, b)$ and $\bullet\, a, \circ\, b$ for combined ownership $(\mathsf{ex}(a), b)$.

The frame-preserving update involves the notion of a *local update*:

Definition 23. *It is possible to do a* local update *from a_1 and b_1 to a_2 and b_2, written $(a_1, b_1) \overset{\mathrm{l}}{\rightsquigarrow} (a_2, b_2)$, if*

$$\forall \alpha, a_{\mathrm{f}}^?.\, \alpha \in \mathcal{V}(a_1) \land a_1 \overset{\alpha}{=} b_1 \cdot a_{\mathrm{f}}^? \Rightarrow \alpha \in \mathcal{V}(a_2) \land a_2 \overset{\alpha}{=} b_2 \cdot a_{\mathrm{f}}^?$$

In other words, the idea is that for every possible frame $a_{\mathrm{f}}^?$ completing b_1 to a_1, the same frame also completes b_2 to a_2.

We then obtain

$$\textsc{auth-update}\ \frac{(a_1, b_1) \overset{\mathrm{l}}{\rightsquigarrow} (a_2, b_2)}{\bullet\, a_1, \circ\, b_1 \rightsquigarrow \bullet\, a_2, \circ\, b_2}$$

3.5.12 Natural Numbers Camera

We can define an RA structure on top of $\mathbb{N}$, with the RA operation being given by addition. This will be particularly useful in conjunction with timecredits later on.

$$\begin{aligned} \overline{\mathcal{V}}(n) &\triangleq \top \\ n_1 \cdot n_2 &\triangleq n_1 + n_2 \\ |n| &\triangleq 0 \end{aligned}$$

The inclusion $\preccurlyeq$ is just the usual $\leq$ relation. Of course, 0 is the unit of the induced discrete camera.

3.5.13 Ordinal Camera

A very similar RA structure can be defined on top of the non-constructive ordinals of §2.2. For addition, we do not use the naive (non-commutative) addition, but instead the (natural) Hessenberg sum which is both associative and commutative. We will denote it by the operation $\oplus : \mathsf{Ord} \to \mathsf{Ord} \to \mathsf{Ord}$.

$$\begin{aligned} \overline{\mathcal{V}}(\alpha) &\triangleq \top \\ \alpha_1 \cdot \alpha_2 &\triangleq \alpha_1 \oplus \alpha_2 \\ |\alpha| &\triangleq 0_{\mathsf{Ord}} \end{aligned}$$

Clearly, 0_{Ord} is the unit of the induced discrete camera. The implication $\alpha_1 \preccurlyeq \alpha_2 \Rightarrow \alpha_1 \preceq \alpha_2$ is valid.

4 Base Logic

Compared to finite Iris, the changes to the base logic are modest. Below we define a transfinite version of uniform predicates and use the existential property to develop new kinds of soundness proofs.

4.1 Uniform Predicates

Given a camera M, we define the COFE $\mathit{UPred}(M)$ of *uniform predicates* over M as follows:

$$M \xrightarrow{\text{mon,ne}} \mathsf{sProp} \triangleq \Sigma \Phi : M \xrightarrow{\text{ne}} \mathsf{sProp}.\, \forall \alpha, a, b.\, a \overset{\alpha}{\preccurlyeq} b \Rightarrow \Phi(a) \overset{\alpha}{\subseteq} \Phi(b)$$

$$\mathit{UPred}(M) \triangleq (M \xrightarrow{\text{mon,ne}} \mathsf{sProp})/_{\equiv}$$

$$\Phi \equiv \Psi \triangleq \forall \beta, a.\, \beta \in \mathcal{V}(a) \Rightarrow (\beta \in \Phi(a) \iff \beta \in \Psi(a))$$

$$\Phi \overset{\alpha}{\equiv} \Psi \triangleq \forall \beta \prec \alpha, a.\, \beta \in \mathcal{V}(a) \Rightarrow (\beta \in \Phi(a) \iff \beta \in \Psi(a))$$

The reader can think of uniform predicates as monotone, step-indexed predicates over a camera that "ignore" invalid elements (as defined by the quotient).

$\mathit{UPred}(-)$ is a locally non-expansive functor from **Camera** to **COFE**.

Given an OFE T, an type A, uniform predicates $P, Q : \mathit{UPred}(M)$ and a predicate $\phi : A \to \mathit{UPred}(M)$, we define the following logical connectives on uniform predicates:

$$t =_T u \triangleq \_ \mapsto \alpha \mapsto t \overset{\alpha}{\equiv} u$$

$$\mathsf{False} \triangleq \_ \mapsto \_ \mapsto \mathsf{False}$$

$$\mathsf{True} \triangleq \_ \mapsto \_ \mapsto \mathsf{True}$$

$$P \wedge Q \triangleq a \mapsto \alpha \mapsto P\, a\, \alpha \wedge Q\, a\, \alpha$$

$$P \vee Q \triangleq a \mapsto \alpha \mapsto P\, a\, \alpha \vee Q\, a\, \alpha$$

$$P \Rightarrow Q \triangleq a \mapsto \alpha \mapsto \forall \beta, b.\, \beta \preceq \alpha \wedge a \preccurlyeq b \wedge \beta \in \mathcal{V}(b) \Rightarrow P\, a\, \beta \Rightarrow Q\, a\, \beta$$

$$\forall x : A.\, \phi x \triangleq a \mapsto \alpha \mapsto \forall x : A.\, (\phi x)\, a\, \alpha$$

$$\exists x : A.\, \phi x \triangleq a \mapsto \alpha \mapsto \exists x : A.\, (\phi x)\, a\, \alpha$$

$$P * Q \triangleq a \mapsto \alpha \mapsto \exists b_1, b_2.\, a \overset{\alpha}{\equiv} b_1 \cdot b_2 \wedge P\, b_1\, \alpha \wedge Q\, b_2\, \alpha$$

$$P \mathrel{-\!\!*} Q \triangleq a \mapsto \alpha \mapsto \forall \beta, b.\, \beta \preceq \alpha \wedge \beta \in \mathcal{V}(a \cdot b) \Rightarrow P\, \beta\, b \Rightarrow Q\, \beta\, (a \cdot b)$$

$$\mathsf{Own}\,(b) \triangleq a \mapsto \alpha \mapsto b \overset{\alpha}{\preccurlyeq} a$$

$$\mathcal{V}(b) \triangleq \_ \mapsto \alpha \mapsto \alpha \in \mathcal{V}(b)$$

$$\Box P \triangleq a \mapsto \alpha \mapsto P\, |a|\, \alpha$$

$$\blacksquare P \triangleq \_ \mapsto \alpha \mapsto P\, \varepsilon\, \alpha$$

$$\triangleright P \triangleq a \mapsto \alpha \mapsto \forall \beta \prec \alpha.\, P\, a\, \beta$$

$$\dot{\Rightarrow} P \triangleq a \mapsto \alpha \mapsto \forall \beta, a'.\, \beta \preceq \alpha \wedge \beta \in \mathcal{V}(a \cdot a') \Rightarrow \exists b.\, \beta \in \mathcal{V}(b \cdot a') \wedge P\, b\, \beta$$

4.2 Deduction System

Uniform predicates give rise to a logic $P \vdash Q$, the *base logic*, defined by:

$$P \vdash Q \triangleq \forall \alpha, a.\, \alpha \in \mathcal{V}(a) \Rightarrow P\, a\, \alpha \Rightarrow Q\, a\, \alpha$$

Laws of intuitionistic higher-order logic with equality.

$$\textsc{Asm}\ \ P \vdash P \qquad \textsc{Cut}\ \ \frac{P \vdash Q \qquad Q \vdash R}{P \vdash R} \qquad \textsc{Eq}\ \ \frac{\phi : T \to \mathit{UPred}(M) \qquad P \vdash \phi\, t \qquad P \vdash t =_T u}{P \vdash \phi\, u} \qquad \textsc{Refl}\ \ \mathsf{True} \vdash t =_\tau t$$

$$\bot\text{E}\ \ \mathsf{False} \vdash P \qquad \top\text{I}\ \ P \vdash \mathsf{True} \qquad \wedge\text{I}\ \ \frac{P \vdash Q \qquad P \vdash R}{P \vdash Q \wedge R} \qquad \wedge\text{EL}\ \ \frac{P \vdash Q \wedge R}{P \vdash Q} \qquad \wedge\text{ER}\ \ \frac{P \vdash Q \wedge R}{P \vdash R} \qquad \vee\text{IL}\ \ \frac{P \vdash Q}{P \vdash Q \vee R}$$

$$\vee\text{IR}\ \ \frac{P \vdash R}{P \vdash Q \vee R} \qquad \vee\text{E}\ \ \frac{P \vdash R \qquad Q \vdash R}{P \vee Q \vdash R} \qquad \Rightarrow\text{I}\ \ \frac{P \wedge Q \vdash R}{P \vdash Q \Rightarrow R} \qquad \Rightarrow\text{E}\ \ \frac{P \vdash Q \Rightarrow R \qquad P \vdash Q}{P \vdash R}$$

$$\forall\text{I}\ \ \frac{\phi : A \to \mathit{UPred}(M) \qquad \forall x : A.\, P \vdash \phi\, x}{P \vdash \forall x : A.\, \phi\, x} \qquad \forall\text{E}\ \ \frac{\phi : A \to \mathit{UPred}(M) \qquad P \vdash \forall x : A.\, \phi\, x \qquad x : A}{P \vdash \phi\, x}$$

$$\exists\text{I}\ \ \frac{\phi : A \to \mathit{UPred}(M) \qquad P \vdash \phi\, x \qquad x : A}{P \vdash \exists x : A.\phi\, x} \qquad \exists\text{E}\ \ \frac{\forall x : A.\, \phi x \vdash Q \qquad \phi : A \to \mathit{UPred}(M)}{\exists x : A.\, \phi x \vdash Q}$$

Laws of (affine) bunched implications.

$$\mathsf{True} * P \dashv\vdash P \qquad P * Q \ \vdash\ Q * P \qquad (P * Q) * R \ \vdash\ P * (Q * R)$$

$$*\text{-MONO}\ \ \frac{P_1 \vdash Q_1 \qquad P_2 \vdash Q_2}{P_1 * P_2 \vdash Q_1 * Q_2} \qquad -\!\!*\text{I-E}\ \ \frac{P * Q \vdash R}{P \vdash Q \mathrel{-\!\!*} R}$$

Laws for the plainness modality.

$$\blacksquare\text{-MONO}\ \ \frac{P \vdash Q}{\blacksquare P \vdash \blacksquare Q} \qquad \blacksquare\text{-E}\ \ \blacksquare P \vdash \square P$$

$$(\blacksquare P \Rightarrow \blacksquare Q) \vdash \blacksquare(\blacksquare P \Rightarrow Q) \qquad \blacksquare((P \Rightarrow Q) \wedge (Q \Rightarrow P)) \vdash P =_{\mathsf{iProp}} Q$$

$$\blacksquare P \vdash \blacksquare\blacksquare P \qquad \forall x.\ \blacksquare P \vdash \blacksquare \forall x.\, P \qquad \blacksquare \exists x.\, P \vdash \exists x.\ \blacksquare P$$

Laws for the persistence modality.

$$\square\text{-MONO}\ \ \frac{P \vdash Q}{\square P \vdash \square Q} \qquad \square\text{-E}\ \ \square P \vdash P$$

$$(\blacksquare P \Rightarrow \square Q) \vdash \square(\blacksquare P \Rightarrow Q) \qquad \square P \wedge Q \vdash \square P * Q$$

$$\square P \vdash \square\square P \qquad \forall x.\ \square P \vdash \square \forall x.\, P \qquad \square \exists x.\, P \vdash \exists x.\ \square P$$

Laws for the later modality.

$$\triangleright\text{-MONO}\ \frac{P \vdash Q}{\triangleright P \vdash \triangleright Q} \qquad \triangleright\text{-I}\ \ P \vdash \triangleright P \qquad \forall x.\ \triangleright P \vdash \triangleright \forall x.\ P \qquad \triangleright P \vdash \triangleright \mathsf{False} \lor (\triangleright \mathsf{False} \Rightarrow P)$$

$$\triangleright P * \triangleright Q \ \vdash\ \triangleright (P * Q) \qquad \square \triangleright P \dashv\vdash \triangleright \square P \qquad \blacksquare \triangleright P \dashv\vdash \triangleright \blacksquare P$$

$$\frac{\mathcal{I} \text{ enjoys the bounded existential property}}{\triangleright \exists x.\ P \vdash \triangleright \mathsf{False} \lor \exists x.\ \triangleright P} \qquad \frac{\mathcal{I} \text{ enjoys the bounded existential property}}{\triangleright (P * Q) \vdash \triangleright P * \triangleright Q}$$

$$\frac{\mathcal{I} \text{ enjoys the finite existential property}}{\triangleright (P \lor Q) \vdash \triangleright P \lor \triangleright Q}$$

Laws for resources and validity.

$$\mathsf{Own}\,(a) * \mathsf{Own}\,(b) \dashv\vdash \mathsf{Own}\,(a \cdot b) \qquad \mathsf{Own}\,(a) \vdash \mathcal{V}(a)$$
$$\mathsf{Own}\,(a) \ \vdash\ \square\,\mathsf{Own}\,(|a|) \qquad \mathcal{V}(a \cdot b) \vdash \mathcal{V}(a)$$
$$\mathsf{True} \ \vdash\ \mathsf{Own}\,(\varepsilon) \qquad \mathcal{V}(a) \vdash \square\,\mathcal{V}(a)$$

$$\frac{\mathcal{I} \text{ enjoys the bounded existential property}}{\triangleright\,\mathsf{Own}\,(a) \vdash \exists b.\ \mathsf{Own}\,(b) \land \triangleright (a = b)}$$

Laws for the basic update modality.

$$\textsc{upd-mono}\ \frac{P \vdash Q}{\dot{\Rightarrow} P \vdash \dot{\Rightarrow} Q} \qquad \textsc{upd-intro}\ \ P \vdash \dot{\Rightarrow} P \qquad \textsc{upd-trans}\ \ \dot{\Rightarrow}\dot{\Rightarrow} P \vdash \dot{\Rightarrow} P \qquad \textsc{upd-frame}\ \ Q * \dot{\Rightarrow} P \vdash \dot{\Rightarrow}(Q * P)$$

$$\textsc{upd-update}\ \frac{a \rightsquigarrow P}{\mathsf{Own}\,(a) \vdash \dot{\Rightarrow} \exists b.\ P\ b \land \mathsf{Own}\,(b)} \qquad \textsc{upd-plainly}\ \ \dot{\Rightarrow} \blacksquare P \vdash P$$

The premise in UPD-UPDATE is a *meta-level* side-condition that has to be proven about a and P.

4.3 Loss of Later Commuting Rules

When using transfinite step-index types (which necessarily refute the bounded existential property, as proved in Section 2), we lose the following rules compared to finite Iris:

(1) the later modality commutes with separating conjunction: $\triangleright (P * Q) \vdash \triangleright P * \triangleright Q$

(2) for infinite types X, the later modality commutes with existential quantification: $\triangleright (\exists x : X.\ P) \vdash \exists x : X.\ \triangleright P$

(3) the later modality can be pulled down under ownership: $\triangleright\,\mathsf{Own}\,(a) \vdash \exists b.\ \mathsf{Own}\,(b) \land \triangleright (a = b)$

In finite Iris, rule (3) is merely used to derive another rule which is still directly provable in the model for transfinite index types, so that its loss is negligible. On the other hand, the loss of (1) and (2) is more serious: these commuting rules are frequently used when opening invariants (which provide the invariant under a later) in the Iris program logic.

However, we can in fact prove that there is no model validating both the existential property, the usual rules for later necessary in a step-indexed logic, and (2).

Lemma 2. *There is no sound step-indexed logic* $\vdash$ *which has*

- *a later operation which is sound in the sense that* $\vdash \triangleright P$ *implies* $\vdash P$*,*
- *Löb induction (if* $\triangleright P \vdash P$*, then* $\vdash P$*),*
- *later existential commuting (2),*
- *and the existential property.*

Proof. We derive a contradiction by assuming that the first three assumptions hold for the logic and refuting the existential property for $\mathbb{N}$.

We claim that there exists $n : \mathbb{N}$ such that $\vdash \triangleright^n \mathsf{False}$. By applying the existential property, we can prove $\exists n : \mathbb{N}.\ \triangleright^n \mathsf{False}$ in the logic.

By Löb induction we may assume $\triangleright \exists n : \mathbb{N}.\ \triangleright^n \mathsf{False}$. Now we use later existential commuting (2) and have $\exists n : \mathbb{N}.\ \triangleright\triangleright^n \mathsf{False}$. We obtain $n : \mathbb{N}$ with $\triangleright^{1+n} \mathsf{False}$. We may pick $n + 1$ as the witness and are done.

It remains to show that $\vdash \triangleright^n \mathsf{False}$ contradicts soundness, *i.e.*, we prove $\vdash \mathsf{False}$. We do an induction on n. In the base case, the claim is trivial. In the successor case, we use the assumed soundness of later and are done. □

Since separating conjunction is defined using existential quantification for splitting the resources in our model, it is not very surprising that our model also does not validate it.

4.4 Soundness

Traditionally, the soundness result of the base logic of Iris is stated as consistency:

$$\mathsf{True} \not\vdash (\triangleright)^n \mathsf{False}$$

where $(\triangleright)^n$ is short for $\triangleright$ being nested n times. Compared to the standard formulation $\mathsf{True} \not\vdash \mathsf{False}$ of consistency, the idea is that it should be impossible to derive a contradiction below the modalities. For $\Box$ and $\blacksquare$, this follows from the elimination rules. For updates, we use the fact that $\dot{\Rightarrow} \mathsf{False} \vdash \dot{\Rightarrow} \blacksquare \mathsf{False} \vdash \mathsf{False}$. However, there is no elimination rule for $\triangleright$, so we declare that it is impossible to derive a contradiction below any number of laters. The traditional soundness theorem does not cover $\mathsf{True} \not\vdash \exists n.(\triangleright)^n \mathsf{False}$ as $\mathsf{True} \vdash \exists n.(\triangleright)^n \mathsf{False}$ is provable in finite Iris.

4.4.1 Satisfiability and Validity

In the following, we pursue a different approach to proving soundness and adequacy results. Instead of stating a single consistency theorem, we develop a modular framework for proving soundness and adequacy results. Given the framework, it is trivial to derive $\mathsf{True} \not\vdash (\triangleright)^n \mathsf{False}$, for example, as a corollary.

To this end, we introduce the *meta-level* notions of *valid* and *satisfiable* propositions. A uniform predicate $P : \mathit{UPred}(M)$ is said to be *valid*, if it is true for all step-indices and valid resources. It is said to be *satisfiable*, if for every step-index there is some valid resource for which the proposition is true.

$$\mathsf{valid}(P) \triangleq \forall \alpha, a.\ \alpha \in \mathcal{V}(a) \Rightarrow P\, a\, \alpha$$
$$\mathsf{satisfiable}(P) \triangleq \forall \alpha.\ \exists a.\ \alpha \in \mathcal{V}(a) \wedge P\, a\, \alpha$$

Lemma 6. *Let ϕ be a meta level proposition embedded into Iris. If* $\mathsf{valid}(\varphi)$ *or* $\mathsf{satisfiable}(\varphi)$ *where φ : Prop, then φ holds.*

Satisfiability vs. validity. Any valid proposition is satisfiable, but not vice versa. Satisfiable propositions are valid, if they do not care about the resource, i.e. they are plain. Validity can be expressed internally in the logic as $\mathsf{True} \vdash P$ whereas we are not aware of an internal notion of satisfiability.

$$\frac{\mathsf{valid}(P)}{\mathsf{satisfiable}(P)} \qquad \frac{\mathsf{satisfiable}(\blacksquare\, P)}{\mathsf{valid}(P)} \qquad \mathsf{valid}(P) \iff \mathsf{True} \vdash P$$

Satisfiability and validity share many of the same properties[10]. Let $\mathsf{pred} \in \{\mathsf{satisfiable}, \mathsf{valid}\}$.

$$\frac{\mathsf{pred}(P) \qquad P \vdash Q}{\mathsf{pred}(Q)} \qquad \frac{\mathsf{pred}(\triangleright\, P)}{\mathsf{pred}(P)}$$

$$\frac{\mathsf{pred}(P \vee Q) \qquad \mathcal{I} \text{ enjoys the finite existential property}}{\mathsf{pred}(P) \vee \mathsf{pred}(Q)}$$

$$\frac{\mathsf{pred}(\exists x : A.\ \phi\, x) \qquad \mathcal{I} \text{ enjoys the small existential property} \qquad A \text{ is small}}{\exists x : A.\ \mathsf{pred}(\phi\, x)}$$

Satisfiability admits an elimination rule for updates on the global resource:

$$\frac{\mathsf{satisfiable}(\dot{\Rightarrow} P)}{\mathsf{satisfiable}(P)}$$

Validity does not admit this rule since it requires the predicate P to hold for all global resources. For example, the update might allocate some resource which is then required to be contained in the global resource by P. Thus, the proposition P would not be valid as it is not true with the empty resource ε.

Validity admits rules for combining uniform predicates such as:

$$\frac{\mathsf{valid}(P) \qquad \mathsf{valid}(Q)}{\mathsf{valid}(P * Q)}$$

which can be derived from the rules above. Combining satisfiable propositions is usually harder since they might be satisfied by different global resources.

[10] We have only mechanized the properties of satisfiable since those are the properties we use in adequacy proofs.

4.4.2 Compositional Soundness Results

We demonstrate the use of valid and satisfiable by proving three soundness results, the original formulation of consistency, a stronger notion of consistency which relies on transfinite step-indexing, and an adequacy theorem for a simplified version of the weakest precondition. The examples demonstrate how we can derive soundness results compositionally: we first prove rules along the lines of the rules shown above and then, we derive the desired soundness result by successive application. These specific examples are not mechanized: their purpose is only to illustrate how we obtain compositional adequacy and soundness proofs using the two notions. (We will see actual adequacy proofs with satisfiable in Section 9.3.) For the last two examples, we assume that the step-index $\mathcal{I}$ satisfies the small existential property and hence also the finite existential property.

Traditional consistency. Recall that the traditional formulation of adequacy is $\mathsf{True} \not\vdash (\triangleright)^n \mathsf{False}$. We first prove an elimination rule for the derived modality $(\triangleright)^n$ and obtain the consistency result as a corollary.

Lemma 7. *Let* $\mathsf{pred} \in \{\mathsf{valid}, \mathsf{satisfiable}\}$.

$$\frac{\mathsf{pred}((\triangleright)^n P)}{\mathsf{pred}(P)}$$

Proof. By induction on n. For $n = 0$, the claim is trivial. For $n > 0$, we have by assumption $\mathsf{pred}(\triangleright(\triangleright)^{n-1} P)$ and thus $\mathsf{pred}((\triangleright)^{n-1} P)$. The claim follows by induction. □

Corollary 1 (Traditional Consistency)**.** *True* $\not\vdash (\triangleright)^n$ *False*

Proof. By way of contradiction assume $\mathsf{True} \vdash (\triangleright)^n \mathsf{False}$. Thus $\mathsf{valid}((\triangleright)^n \mathsf{False})$ and hence $\mathsf{valid}(\mathsf{False})$ by Lemma 7, a contradiction by Lemma 6. □

Transfinite consistency. Recall that the traditional consistency theorem does not cover $\mathsf{True} \not\vdash \exists n.(\triangleright)^n \mathsf{False}$ as $\mathsf{True} \vdash \exists n.(\triangleright)^n \mathsf{False}$ is provable in finite Iris. In the transfinite version of Iris, we can also prove this stronger consistency theorem. That is, we can prove $\mathsf{True} \not\vdash \triangleright^\omega \mathsf{False}$ where $\triangleright^\omega P \triangleq \exists n.\ (\triangleright)^n P$.

Lemma 8. *Let* $\mathsf{pred} \in \{\mathsf{valid}, \mathsf{satisfiable}\}$.

$$\frac{\mathsf{pred}(\triangleright^\omega P)}{\mathsf{pred}(P)}$$

Proof. Let $\mathsf{pred}(\triangleright^\omega P)$. Then $\mathsf{pred}(\triangleright^n P)$ for some $n : \mathbb{N}$. The claim follows by Lemma 7. □

Corollary 2 (Transfinite Consistency)**.** *True* $\not\vdash \triangleright^\omega$ *False*

A simplified weakest precondition. We demonstrate the compositionality of the approach by proving adequacy of a simplified version of the weakest precondition. The actual definition of the weakest precondition, which is used in the definition of Hoare triples, can be found in Section 9. We define[11]:

$$\mathsf{wp}\ v\ \{\phi\} \triangleq \phi\, v$$
$$\mathsf{wp}\ e\ \{\phi\} \triangleq (\exists e'.\ e \rightsquigarrow e') \wedge \forall e'.\ e \rightsquigarrow e' \Rightarrow \dot{\Rrightarrow} \triangleright^\omega\, \mathsf{wp}\ e'\ \{\phi\} \quad \text{where } e \text{ is not a value}$$

[11] The modality $\triangleright^\omega$ is contractive which makes it possible to define the weakest precondition using the *fix* operator given by Banach's Theorem 1.

where we assume e, e' range over expressions of some language, v over values of that language, and $e \rightsquigarrow e'$ denotes a single, small step in the operational semantics.

In the following, we prove adequacy of the weakest precondition. That is, we prove that if one can prove $\mathsf{True} \vdash \mathsf{wp}\ e_s\ \{\phi\}$ and further $e_s \rightsquigarrow^* e_t$, then e_t is a value or there is some e_t' such that $e_t \rightsquigarrow e_t'$. In general, we obtain a *compositional* proof of adequacy by proving soundness results for the modalities used inside of the weakest precondition and then using those results to obtain soundness results about the weakest precondition itself. We already have soundness results for $\dot{\Rightarrow} P$ and $\triangleright^\omega P$ which means we can proceed to derive soundness results for the weakest precondition itself.

Lemma 9.

$$\frac{\mathsf{satisfiable}(\mathit{wp}\ e\ \{\phi\}) \qquad e \rightsquigarrow e'}{\mathsf{satisfiable}(\mathit{wp}\ e'\ \{\phi\})}\ \textit{WP-SOUND-STEP} \qquad \frac{\mathsf{satisfiable}(\mathit{wp}\ e\ \{\phi\}) \qquad e \rightsquigarrow^* e'}{\mathsf{satisfiable}(\mathit{wp}\ e'\ \{\phi\})}\ \textit{WP-SOUND-STEPS}$$

$$\frac{\mathsf{satisfiable}(\mathit{wp}\ e\ \{\phi\}) \qquad e \text{ is not a value}}{\mathsf{satisfiable}(\exists e'.\ e \rightsquigarrow e')}\ \textit{WP-SOUND-PROGRESS}$$

Proof.

1. *WP-SOUND-STEP.* Since $e \rightsquigarrow e'$, we know e is not a value. Thus, we have:

   $$\mathsf{satisfiable}((\exists e'.\ e \rightsquigarrow e') \land \forall e'.\ e \rightsquigarrow e' \Rightarrow \dot{\Rightarrow} \triangleright^\omega \mathsf{wp}\ e'\ \{\phi\})$$

   and hence $\mathsf{satisfiable}(\forall e'.\ e \rightsquigarrow e' \Rightarrow \dot{\Rightarrow} \triangleright^\omega \mathsf{wp}\ e'\ \{\phi\})$. We obtain $\mathsf{satisfiable}(\dot{\Rightarrow} \triangleright^\omega \mathsf{wp}\ e'\ \{\phi\})$ by $e \rightsquigarrow e'$ and thus $\mathsf{satisfiable}(\triangleright^\omega \mathsf{wp}\ e'\ \{\phi\})$. Using Lemma 8, we have $\mathsf{satisfiable}(\mathsf{wp}\ e'\ \{\phi\})$.

2. *WP-SOUND-STEPS.* By induction on $e \rightsquigarrow^* e'$.

   (a) Let $e = e'$. The claim is trivial.

   (b) Let $e \rightsquigarrow e'' \rightsquigarrow^* e'$. By WP-SOUND-STEP, we have $\mathsf{satisfiable}(\mathsf{wp}\ e''\ \{\phi\})$. By induction, we obtain $\mathsf{satisfiable}(\mathsf{wp}\ e'\ \{\phi\})$.

3. *WP-SOUND-PROGRESS.* Since e is not a value, we have:

   $$\mathsf{satisfiable}((\exists e'.\ e \rightsquigarrow e') \land \forall e'.\ e \rightsquigarrow e' \Rightarrow \dot{\Rightarrow} \triangleright^\omega \mathsf{wp}\ e'\ \{\phi\})$$

   Thus $\mathsf{satisfiable}(\exists e'.\ e \rightsquigarrow e')$.

□

We can now derive an adequacy theorem for the simplified weakest precondition:

Lemma 10 (Simplified Adequacy)**.** *If* $\mathsf{True} \vdash \mathit{wp}\ e_s\ \{\phi\}$ *and* $e_s \rightsquigarrow^* e_t$*, then* e_t *is a value or there is some* e_t' *such that* $e_t \rightsquigarrow e_t'$*.*

Proof. By assumption, we have $\mathsf{valid}(\mathsf{wp}\ e_s\ \{\phi\})$ and thus $\mathsf{satisfiable}(\mathsf{wp}\ e_s\ \{\phi\})$. By Lemma 9, we have $\mathsf{satisfiable}(\mathsf{wp}\ e_t\ \{\phi\})$. If e_t is a value, the claim follows. If not, then we have $\mathsf{satisfiable}(\exists e_t'.\ e_t \rightsquigarrow e_t')$. Thus, there is some e_t' such that $e_t \rightsquigarrow e_t'$ by Lemma 6. □

5 Recursive Domain Equation

We will use UPred as the basis for a step-indexed model iProp for Transfinite Iris, parameterised over a resource type (formally, a camera). Importantly, in order to support higher-order ghost state, the resource type must be allowed to depend on iProp itself. We therefore formulate the resource type as a function G parameterised over the model and returning a camera. Thus, we need to solve a recursive equation of the following form:

$$\mathit{iProp} \simeq \mathsf{UPred}(\mathit{Res}) \qquad \text{with} \quad \mathit{Res} \triangleq G(\mathit{iProp})$$

Formally, we can interpret G as a bifunctor $G : \mathbf{OFE}^{op} \times \mathbf{OFE} \to \mathbf{UCMRA}$ of mixed variance, where **OFE** is the category of OFEs with non-expansive maps between them and **UCMRA** is the evident category of unital cameras. Composing with the $\mathsf{UPred} : \mathbf{UCMRA} \to \mathbf{COFE}$ functor, where **COFE** is the subcategory of **OFE** containing only the COFEs, our problem reduces to finding a COFE X such that

$$F(X, X) \simeq X$$

for a functor $F : \mathbf{OFE}^{op} \times \mathbf{OFE} \to \mathbf{COFE}$.

F (and G, respectively) has mixed variance to allow for recursive occurrences in negative positions. For instance, the classic domain equation for the untyped λ calculus can be formulated as

$$F(X^-, X^+) = (X^- \to X^+).$$

In practice, we have stronger structural requirements on F, most importantly that it is locally contractive. These will be outlined below.

5.1 Related Work

In Iris, the solution of recursive domain equations was based on a theorem by America and Rutten [America and Rutten, 1989; Birkedal et al., 2010]. This allows to solve domain equations where F is of the form $F : \mathbf{COFE}^{op} \times \mathbf{COFE} \to \mathbf{COFE}$. Compared to the form $\mathbf{OFE}^{op} \times \mathbf{OFE} \to \mathbf{COFE}$ above, this is a weaker requirement on F, as F only needs to work on COFEs. While the construction of the Iris model itself does not require such a stronger theorem, more complex constructions like the one in Actris [Hinrichsen et al., 2019] seem to rely on this.

In contrast to Iris, for Transfinite Iris we need to solve such domain equations over arbitrary step-index types, not just over ω. For COFEs, this is not covered by the literature.

Svendsen et al. [2016] take steps in exploring transfinite logical relations and have to solve domain equations over ω^2 in the process. They claim to solve domain equations of the form $\mathbf{COFE}^{op} \times \mathbf{COFE} \to \mathbf{COFE}$, but hinge on the misconception that non-expansive maps preserve bounded limits, in particular

$$\forall (f : A \xrightarrow{\text{ne}} B)(c : \mathrm{bchain}_\omega(A)).f(\lim_{\gamma \prec \omega} c_\gamma) = \lim_{\gamma \prec \omega} f(c_\gamma),$$

which is not true in general. For instance, consider the definition of UPred (which also satisfies the relevant aspects of their **COFE** definition) over some camera and the constant chain $(\top)_{\gamma \prec \omega}$ with the function $f : \mathsf{UPred} \to \mathsf{UPred} \triangleq \lambda x.x \wedge (\exists n.\, \triangleright^n \bot)$. Then $f(\lim_{\gamma \prec \omega} c_\gamma)$ does not hold at ω, but $\lim_{\gamma \prec \omega} f(c_\gamma)$ does.

The inverse limit that is taken in the case of limit ordinals of the form $\omega \cdot i$ for $i \in \mathbb{N}$ is thus not a COFE, breaking the proof. Luckily, the rest of their paper does also work with a weaker domain equation solver for functors $\mathbf{OFE}^{op} \times \mathbf{OFE} \to \mathbf{COFE}$, as they only need to solve domain equations over (some form of)

UPred, similar to our situation. Even with this stronger requirement, their construction contains another flaw, which we will explain below.

We note that the category-theoretic solution of recursive domain equations in a more general setting over sheaves is well understood. Birkedal et al. [2011] show that for any complete Heyting algebra A with a well-founded basis, domain equations in the topos of sheaves over A can be solved. However, as sheaves are hard to mechanise and it does not seem feasible to use them as a model for Transfinite Iris in Coq, we need the more restrictive setting of COFEs. COFEs over transfinite step-index types are less well-behaved than sheaves, which is cause for some of the problems in Svendsen et al. [2016]'s proof. For instance, the category **COFE** does not have limits, in general.

5.2 A Solution Sketch

Formally, we prove the statement

Theorem 2 (Solution of Transfinite Recursive Domain Equations)**.** *Let $\mathbb{1}$ be the discrete OFE on the unit type. Given a locally contractive bifunctor $F : \mathbf{OFE}^{op} \times \mathbf{OFE} \to \mathbf{COFE}$ which satisfies*

- *$F(\mathbb{1}, \mathbb{1})$ is inhabited by $\star_{F(\mathbb{1},\mathbb{1})}$,*
- *$F(O, O)$ has unique limits of bounded chains up to limit ordinals for any O (**Limit Uniqueness**),*
- *and $F(O, O)$ has a truncation operation for any O (**Truncatability**),*

there exists a COFE T such that $F(T, T) \simeq T$[12]*.*

Note that we require $F : \mathbf{OFE}^{op} \times \mathbf{OFE} \to \mathbf{COFE}$ instead of $\mathbf{COFE}^{op} \times \mathbf{COFE} \to \mathbf{COFE}$. It does not seem possible to go with the weaker requirement in the transfinite case.

The two latter requirements on F can be understood as "regularity conditions" on COFEs. Our definition of COFEs is quite minimal because more regularity is only used for select COFEs in the Coq formalisation in the model construction, but the conditions are useful for solving domain equations. We give their formal statement and an explanation below. Interestingly, they are true of any COFE under classical logic with a choice principle.

We will now give an outline of the construction to motivate the formulation of the theorem and give an intuitive account before delving into the technical details below.

5.2.1 Solution over ω

We start with a recap of the proof over ω. Thus assume a locally contractive bifunctor $F : \mathbf{OFE}^{op} \times \mathbf{OFE} \to \mathbf{COFE}$ such that $F(\mathbb{1}, \mathbb{1})$ is inhabited. Our goal is to define X with $F(X, X) \simeq X$.

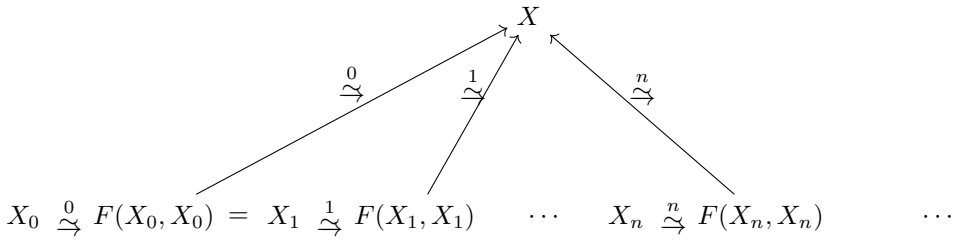
The key idea is to follow the step-index structure and define, for every ordinal up to ω, an approximation which will become "finer" with every step. Formally, for every $i : \mathbb{N}$, a COFE X_i with $X_i \overset{i}{\simeq} F(X_i, X_i)$, is defined by induction. $X_i \overset{i}{\simeq} F(X_i, X_i)$ means that the two COFEs are (asymmetrically) isomorphic up to i, *i.e.,* there are maps $\phi_i : X_i \xrightarrow{\text{ne}} F(X_i, X_i), \psi_i : F(X_i, X_i) \xrightarrow{\text{ne}} X_i$ with $\phi_i \circ \psi_i \overset{i}{=} id$ and $\psi_i \circ \phi_i = id$. This asymmetry does not only make sense intuitively (there should be an injection from "smaller" approximations to "larger" approximations), but it is crucially needed later on.

For the base case, we set $X_0 := F(\mathbb{1}, \mathbb{1})$.

[12] Probably, this COFE is also unique (classically), but we'd need to investigate that.

In the successor case, we already have an approximation X_i with accompanying maps ϕ_i, ψ_i. Intuitively, we now just apply the functor F once, which will bring our approximation closer to a full solution as F is locally contractive. We set $X_{i+1} \triangleq F(X_i, X_i)$, $\phi_{i+1} \triangleq F(\psi_i, \phi_i)$, and $\psi_{i+1} \triangleq F(\phi_i, \psi_i)$. As functors preserve composition, the equations for the isomorphisms are easy to prove.

In the end, we have approximations $X_i \overset{i}{\simeq} F(X_i, X_i)$ for every $i : \mathbb{N}$. In order to define X, we need some form of limit of the X_i. We do not give a precise definition in this outline, but the situation looks like this:



Intuitively, X should subsume every X_i by an isomorphism up to i, as we want to define an isomorphism $X \simeq F(X, X)$ by "lifting" the "bounded" isomorphisms $X_i \overset{i}{\simeq} F(X_i, X_i)$. A first approach to achieving this is to set X as $\prod_{i:\mathbb{N}} X_i$ (with pointwise equality) or equivalently (due to the isomorphisms) as $\prod_{i:\mathbb{N}} F(X_i, X_i)$. Choosing one over the other does not make a fundamental difference, but the latter will make the presentation a bit simpler later on. Defining maps $p_{i,\omega} : X \overset{\text{ne}}{\to} F(X_i, X_i)$ is trivial with this definition as we can just project out. However, this simple choice does not allow us to define inverse maps $F(X_i, X_i) \overset{\text{ne}}{\to} X$, yet, as we somehow need to turn an element $x : F(X_i, X_i)$ into elements of $F(X_j, X_j)$ for every j.

It turns out that we need more structure between the X_i for this. Namely, we can define embedding-projection pairs $e_{i,j} : X_i \overset{\text{ne}}{\to} X_j$ and $p_{i,j} : X_j \overset{\text{ne}}{\to} X_i$ for $i < j$, such that $e_{i,j} \circ p_{i,j} \overset{i}{=} id$ and $p_{i,j} \circ e_{i,j} = id$. The definitions proceed by inductively composing the maps ϕ_i, ψ_i.

With this, we can define maps $e_{i,\omega} : F(X_i, X_i) \overset{\text{ne}}{\to} X$. Proving that $e_{i,\omega}$ inverts $p_{i,\omega}$ is not yet possible, however: when using $p_{i,\omega}$ on $x : X$, we throw away all but the i-th component of x; in order to accurately restore the other components by using the $e_{i,j}, p_{i,j}$ maps, some form of coherence is required for the elements of X. Therefore, we define X by "slicing out" the elements of the product $\prod_{i:\mathbb{N}} F(X_i, X_i)$ for which the components are coherent, *i.e.,* $\forall i, j. p_{i+1,j+1}(x_j) = x_i$. We call this an equalisation property, referring to the underlying category-theoretic notion of equalisers. Then, defining $e_{i,\omega}$ and proving that it is well-defined, *i.e.,* that its image satisfies this coherence, requires that $p_{i,j} \circ e_{i,j} = id$, motivating the asymmetry in our notation $\dot{\simeq}$.

Next, we define $X \simeq F(X, X)$. As we have $X_i \overset{i}{\simeq} X$, we consequently get $F(X_i, X_i) \overset{i}{\simeq} F(X, X)$. For defining $\psi : F(X, X) \overset{\text{ne}}{\to} X$, it is therefore possible to just map down to each component individually. The converse direction $\phi : X \overset{\text{ne}}{\to} F(X, X)$ is more difficult: we can define a map into $F(X, X)$ for each component individually and, for each $x : X$, the resulting sequence of elements in $F(X, X)$ will form a chain, of which we can take the limit (as $F(X, X)$ is a COFE).

This finishes our outline of the essential parts for the ω case.

5.2.2 Additions for Transfinite Index Types

The most obvious change in the transfinite case is that we also need to define intermediate approximations X_β at limit ordinals β. Conceptually, the construction for this is quite similar to the final limit of the

proof for ω, but we use the bounded limit operation for defining the map $X_\beta \xrightarrow{\text{ne}} F(X_\beta, X_\beta)$. A difficulty with proving the bounded isomorpisms here is that bounded limits of chains are potentially sensitive to "differences" of the chain's elements at indices $> \beta$ which are not interesting when defining the β-th approximation. We would like to make such differences irrelevant and eliminate all interesting behaviour above index β. In principle, there are two available choices here:

- we make the limit operations ignore such differences, that is, for any two chains $c, d : \mathrm{bchain}_\alpha$, if $c_\gamma \overset{\gamma}{=} d_\gamma$ for any $\gamma \prec \alpha$, then $\lim_{\gamma \prec \alpha} c_\gamma = \lim_{\gamma \prec \alpha} d_\gamma$ (**strongly unique limits**).
- we completely remove such differences above β at the OFE level, *i.e.,* setup X_β such that for all $x, y : X_\beta, x \overset{\beta}{=} y \Leftrightarrow x = y$ (**truncate** X_β at index β).

The first approach is taken by Svendsen et al. [2016]. Their proof goes wrong, however, as they again rely on non-expansive maps preserving bounded limits in setting up ϕ, ψ in the limit case. It does not seem clear how to fix this, except for going with the second approach (*truncation*) which eliminates unwanted differences at a much more global level. Thus, we take the second approach.

Another difficulty lies with the fact that COFEs now need to be equipped with bounded limit operations. It turns out that this is quite problematic for "slicing out" elements of a COFE, for instance by forming dependent sums. In practice, it seems like the best we can achieve is to make the limit approximation X_β a "lower-bounded COFE" in the following sense: it only has limit operations for chains which go at least up to i. If a chain is too short, then it does not seem possible to prove that that the limits of chains whose elements live in X_β are again in X_β, satisfying the equalisation property. Svendsen et al. [2016] wrongly used that non-expansive maps preserve bounded limits here. However, by restricting to functors $F : \mathbf{OFE}^{op} \times \mathbf{OFE} \to \mathbf{COFE}$, we can just turn the OFE X_i into a COFE again by applying F to it.

Finally, the structure of the induction becomes considerably more complicated: as we need to have the maps $e_{i,j}, p_{i,j}$ defined to use the inverse limit construction in the limit case and to even state the type of $e_{i,j}$, we need to know X_i and X_j, these must be defined *simultaneously* to the types X_i. In contrast, for the ω proof this is not necessary as the inverse limit only has to be taken once at the very end and thus this can be split up over multiple recursions. Simultaneous definitions of types and maps/properties on them are possible in enclosing type theories featuring induction-recursion principles [Dybjer, 2000]. In type theories without native support for this, considerable effort has to be taken to encode "small" induction-recursion[13] [Hancock et al., 2013].

5.3 Preliminaries

In the following, we setup some basic definitions and prove important lemmas before delving into the details of the proof. Let us fix a step-index type $\mathcal{I}$ for the rest of this section.

We use the notation $A \overset{\alpha}{\simeq} B$ to denote that two OFEs A, B are isomorphic up to α, *i.e.,* there are non-expansive maps $f : A \xrightarrow{\text{ne}} B, g : B \to A$ which invert each other up to α, meaning $f \circ g \overset{\alpha}{=} id$ and $g \circ f \overset{\alpha}{=} id$. If the stronger equality $g \circ f = id$ holds, then we write $A \overset{\alpha}{\simeq\!\!\to} B$.

Whenever an equality in an OFE A holds for all indices $\prec \beta$, we use the notation

$$a \overset{<\beta}{=} b \triangleq \forall \alpha \prec \beta. a \overset{\alpha}{=} b.$$

Throughout this section, we make use of the usual notations for limits. If $c : \mathrm{chain}(A)$ for a COFE A, then we write $\lim_\alpha c_\alpha$ for $\lim(c)$. If $c : \mathrm{bchain}_\beta(A)$, then we write $\lim_{\alpha \prec \beta} c_\alpha$ for $\mathrm{blim}(c)$.

[13] Small induction-recursion does not allow the definition of universes and is all what is needed by our construction.

Fact 1 (Non-expansive maps preserve bounded limits *in a restricted way*). *Suppose that* $f : A \xrightarrow{ne} B$ *and* $c : \mathrm{bchain}_\beta(A)$. *Then*

$$f(\lim_{\gamma \prec \beta} c_\gamma) \overset{\leq \beta}{=} \lim_{\gamma \prec \beta}(f(c_\gamma)).$$

In Section 5.1, we have already seen a counterexample showing that a stronger equality need not hold.

Fact 2 (Non-expansive maps preserve limits). *Suppose that* $f : A \xrightarrow{ne} B$ *and* $c : \mathrm{chain}(A)$. *Then*

$$f(\lim_{\gamma} c_\gamma) = \lim_{\gamma} f(c_\gamma).$$

Fact 3 (Weak Limit Uniqueness). *Given two bounded chains* $c, d : \mathrm{bchain}_\beta(A)$ *such that* $\forall \gamma \prec \beta, c_\gamma \overset{\gamma}{=} d_\gamma$, *we have that* $\lim_{\gamma \prec \beta} c_\gamma \overset{\leq \beta}{=} \lim_{\gamma \prec \beta} d_\gamma$.

Fact 4 (Limit uniqueness for full chains). *Given two chains* $c, d : \mathrm{chain}(A)$ *such that* $\forall \gamma, c_\gamma \overset{\gamma}{=} d_\gamma$, *we have that* $\lim_\gamma c_\gamma = \lim_\gamma d_\gamma$.

Note how these facts differ in strength between full limits and bounded limits.

Definition 4 (Truncation). An OFE A is truncated at an ordinal α if it has no interesting behaviour after ordinal α. Formally,

$$\mathrm{truncated}_\alpha(A) \triangleq \forall x y \beta. x \overset{\beta}{=} y \leftrightarrow x \overset{\min(\alpha,\beta)}{=} y.$$

This means that equality at $\beta \succeq \alpha$ is just equality at α.

5.4 Additional Properties of OFEs

In this subsection, we motivate the Truncatability and Limit Uniqueness requirements of Theorem 2. It would have been possible to choose other requirements, one of which (**Strong Limit Uniqueness**) is taken in Svendsen et al. [2016]; however, as noted in Section 5.2.2, truncation seems to be the right choice. We (constructively) prove relations between them, but classically with choice, all of these properties are trivial.

Limit Uniqueness (A1) The bounded limit of every chain $c : \mathrm{bchain}_\beta(A)$ is unique up to $\overset{\beta}{=}$ with this property for every ordinal β. Formally, if $c, d : \mathrm{bchain}_\beta(A)$ and $c_\gamma \overset{\gamma}{=} d_\gamma$ for all $\gamma \prec \beta$, then $\lim_{\gamma \prec \beta} c_\gamma \overset{\beta}{=} \lim_{\gamma \prec \beta} d_\gamma$.

One can consider a (weaker) variation (**A1'**) where we add the requirement that β is a limit ordinal.

Strong Limit Uniqueness (A2) One can consider a variation of (A1) where full equality is required: if $c, d : \mathrm{bchain}_\beta(A)$ and $c_\gamma \overset{\gamma}{=} d_\gamma$ for all $\gamma \prec \beta$ and β is a limit ordinal, then $\lim_{\gamma \prec \beta} c_\gamma = \lim_{\gamma \prec \beta} d_\gamma$.

Intuitively, this means that the limit operation chooses a unique representative of chains for each equivalence class of the pointwise equality relation and then takes the limit of this representative.

Truncatability (A3) Intuitively, an OFE A is truncatable if it allows to cut off all behaviour of elements after a certain ordinal. For that, it offers a collection of OFEs $[A]_\alpha$ such that $A \overset{\alpha}{\simeq} [A]_\alpha$ and $\mathrm{truncated}_\alpha [A]_\alpha$ for every ordinal α. We use $\lceil \cdot \rceil^\alpha : [A]_\alpha \xrightarrow{\mathrm{ne}} A$ and $\lfloor \cdot \rfloor_\alpha : A \xrightarrow{\mathrm{ne}} [A]_\alpha$ to denote the two witnessing maps.

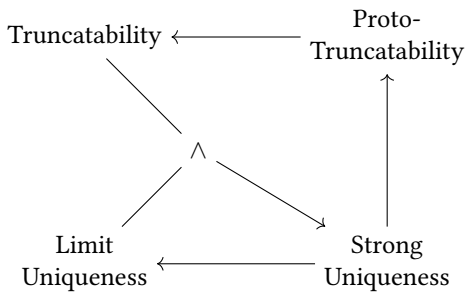


Figure 2: Constructively provable relations between the OFE properties.

Proto-Truncatability (A4) This property follows the same intuition as **(A3)**, but is more elementary. An OFE A is *proto-truncatable* iff there is a (non-expansive) operation $\mathsf{trunc}_\alpha : A \xrightarrow{\text{ne}} A$ for every ordinal α. We require that, if $x \overset{\alpha}{=} y$, then $\mathsf{trunc}_\alpha x = \mathsf{trunc}_\alpha y$. Moreover, $x \overset{\alpha}{=} \mathsf{trunc}_\alpha x$.

Intuitively, trunc implements a choice operation selecting a unique representative for each equivalence class of $\overset{\alpha}{=}$.

We have the following relations (depicted in Figure 2) between these properties:

Lemma 3 (Strong Limit Uniquness implies Proto-Truncatability)**.** *Let X be a COFE satisfying **(A2)**. Then X satisfies **(A4)**.*

Proof. The idea is to take the limit of a constant chain for the truncation operation:

$$\mathsf{trunc}_\alpha(x) \triangleq \lim_{\gamma \prec \alpha} x$$

- trunc_α is non-expansive: suppose that $x \overset{\beta}{=} y$. By COFE-BCOMPL-NE, $\lim_{\gamma\prec\alpha} x \overset{\beta}{=} \lim_{\gamma\prec\alpha} y$.
- Suppose that $x \overset{\alpha}{=} y$. Then in particular for all $\gamma \prec \alpha$ it holds that $x \overset{\gamma}{=} y$, so that by **(A2)** $\lim_{\gamma\prec\alpha} x = \lim_{\gamma\prec\alpha} y$.
- $x \overset{\alpha}{=} \lim_{\gamma\prec\alpha} x$ holds by definition of limits.

□

Lemma 4 (Proto-Truncatability implies Truncatability)**.** *Let A by an OFE satisfying **(A4)**. Then A satisfies **(A3)**.*

Proof. Fix a step-index α. Define

$$\begin{aligned}
\overline{A} &\triangleq \overline{x} \quad \text{for} \quad x : X \\
[A]_\alpha &\triangleq \overline{A} \quad \text{with} \quad \overline{x} \overset{\beta}{=} \overline{y} \triangleq \mathsf{trunc}_\alpha x \overset{\beta}{=} \mathsf{trunc}_\alpha y \\
\lfloor x \rfloor_\alpha &\triangleq \overline{x} \\
\lceil \overline{x} \rceil^\alpha &\triangleq \mathsf{trunc}_\alpha x
\end{aligned}$$

Here, we have used $\overline{A}$ and $\overline{x}$ to make clear that these are (essentially) the same types, but the equivalence relations on top change. The most interesting part is verifying that the truncations have limits again. This can be done by mapping a chain in $[A]_\alpha$ to a chain in A, taking the limit there, and mapping back. □

Lemma 5 (Truncatability and Limit Uniqueness imply Strong Limit Uniqueness)**.** *Assume that X is a COFE satisfying **(A1)** and **(A3)**. Then X satisfies **(A2)**.*

Proof. The key idea is to redefine the bounded limit operation. For bounded limits up to β, we first map the chain into $[X]_\beta$, take the limit there, apply **(A1)**, and then map back. This yields a unique choice of limits due to the normalisation performed by the truncation, following directly the intuition for strongly unique limits given above: we first choose an essentially unique representative for chains and then take a limit. □

Lemma 6. *Let X be a OFE. Assuming* XM, PE, FE, *and* Choice, *X satisfies property **(A4)**.*
*Moreover, if X is a COFE, it satisfies **(A2)** under these axioms.*

We also note that the non-expansiveness condition on bounded limits COFE-BCOMPL-NE is already implied by Strong Limit Uniqueness.

Strong Limit Uniqueness is required of COFEs in the ω^2 proof by Svendsen et al. [2016]. In principle, our assumptions Limit Uniqueness and Truncatability are equivalent to theirs, but we have additionally weakened Limit Uniqueness **(A1)** to only require the uniqueness for chains up to limit ordinals **(A1')**. However, the strongly unique limits effectively need to be used to regularly truncate approximations to the solution in the proof. Svendsen et al. [2016] use Strongly Unique Limits differently, which is cause for a flaw in their limit case construction.

More on Truncation Consider again the "truncatability" offered by property (**A3**). We call $[A]_\alpha$ the truncation of A at α, for an OFE A. Recall that we require $A \overset{\alpha}{\simeq} [A]_\alpha$.

If A is a COFE, then every truncation $[A]_\alpha$ is also a COFE, and if A satisfies property **(A1)**, **(A1')**, or **(A2)**, then also $[A]_\alpha$ satisfies it.

We can let maps between two truncatable OFEs A, B descend to arbitrary truncations. Let $f : A \xrightarrow{\text{ne}} B$ and $\alpha, \beta : \mathcal{I}$, then define $[f]^\alpha_\beta : [A]_\alpha \xrightarrow{\text{ne}} [B]_\beta$ by

$$[f]^\alpha_\beta(x) \triangleq \lfloor f(\lceil x \rceil^\alpha) \rfloor_\beta.$$

Note that this map is again non-expansive as it is the composition of non-expansive maps.

Truncation of maps is functorial in the following sense: if $f : A \xrightarrow{\text{ne}} B$, $g : B \xrightarrow{\text{ne}} C$, and $\gamma_0, \gamma_1, \gamma_2 : \mathcal{I}$, then $[g \circ f]^{\gamma_2}_{\gamma_0} \overset{\gamma_1}{=} [g]^{\gamma_1}_{\gamma_0} \circ [f]^{\gamma_2}_{\gamma_1}$.

It may seem peculiar that we see truncatability as an explicit property of a particular OFE. The problem is that general (constructive) constructions like quotienting in a suitable way do not yield the properties we need: the expansion operation $\lceil \cdot \rceil^\alpha$ will not be non-expansive in this case – we really need that the truncations choose canonical representatives. Intuitively, one can understand this as follows: with a truncation at α by quotienting, we just hide the behaviour of elements at larger ordinals than α. What we really need, however, is to cut off all ways to distinguish elements at ordinals larger than α, so that expanding/mapping out of the truncation cannot restore differences at ordinals larger than α that were not already there at α.

5.5 Proof

We now turn to the actual proof. Fix a locally contractive functor $F : \mathbf{OFE}^{\mathrm{op}} \times \mathbf{OFE} \to \mathbf{COFE}$. Moreover, as mentioned previously, we require that for every $X : \mathbf{OFE}$:

- $F(X, X)$ is truncatable at all ordinals **(A3)**
- $F(X, X)$ has unique limits at limit ordinals **(A1')**

Finally, we need that $F(\mathbb{1}, \mathbb{1})$ is inhabited by an element $\star_{F(\mathbb{1},\mathbb{1})}$.

We show that there exists an (inhabited) COFE X such that $X \simeq F(X, X)$. Our proof roughly follows the outline in Section 5.2, but adds in all the formal details needed for truncation, verifying the equations, and making it a sound induction.

Essentially, the proof is by recursion on the well-founded order of the ordinal type, in a scheme which is akin to small induction-recursion. In each step β of the induction, we define an approximation X_β which satisfies the isomorphism we are after up to β and is truncated at β. Compared to the previous outline, we truncate each approximation X_β at β in order to avoid the mentioned difficulties in the limit case. All of these approximations will need to form a chain of which we will take the limit in the end; therefore, we need additionally embedding-projection pairs $e_{\gamma,\beta} : X_\gamma \xrightarrow{\text{ne}} X_\beta$ and $p_{\gamma,\beta} : X_\beta \xrightarrow{\text{ne}} X_\gamma$ for every $\gamma \prec \beta$. These embeddings and projections need to be inverses up to β and moreover functorial.

The embeddings and projections need to be defined in the same induction as the approximations X_β themselves as we need them to define the approximation in the limit case – therefore, we need to take an induction-recursion-like approach.

In total, we define the following things in each step at ordinal β, all packed up in a dependent tuple:

$$X_\beta : \mathbf{COFE} \tag{IH-0}$$
$$\forall \gamma \prec \beta . e_{\gamma,\beta} : X_\gamma \xrightarrow{\text{ne}} X_\beta \tag{IH-1}$$
$$\forall \gamma \prec \beta . p_{\gamma,\beta} : X_\beta \xrightarrow{\text{ne}} X_\gamma \tag{IH-2}$$
$$p_{\gamma,\beta} \circ e_{\gamma,\beta} = id \tag{IH-3}$$
$$e_{\gamma,\beta} \circ p_{\gamma,\beta} \overset{\gamma}{=} id \tag{IH-4}$$
$$\forall \gamma_0 \prec \gamma_1 \prec \gamma_2 \preceq \beta . e_{\gamma_0,\gamma_2} = e_{\gamma_1,\gamma_2} \circ e_{\gamma_0,\gamma_1} \tag{IH-5}$$
$$\forall \gamma_0 \prec \gamma_1 \prec \gamma_2 \preceq \beta . p_{\gamma_0,\gamma_2} = p_{\gamma_0,\gamma_1} \circ p_{\gamma_1,\gamma_2} \tag{IH-6}$$
$$\phi_\beta : X_\beta \xrightarrow{\text{ne}} [F(X_\beta, X_\beta)]_{1+\beta} \tag{IH-7}$$
$$\psi_\beta : [F(X_\beta, X_\beta)]_{1+\beta} \xrightarrow{\text{ne}} X_\beta \tag{IH-8}$$
$$\psi_\beta \circ \phi_\beta = id \tag{IH-9}$$
$$\phi_\beta \circ \psi_\beta \overset{\beta}{=} id \tag{IH-10}$$
$$\text{truncated}_\beta(X_\beta) \tag{IH-11}$$

If β is a successor ordinal $\beta = 1 + \beta'$, then (IH-12)

$$X_{1+\beta'} = [F(X_{\beta'}, X_{\beta'})]_{1+\beta'} \quad \text{(IH-12-I)}$$

$$\phi_{1+\beta'} = [F(\psi_{\beta'}, \phi_{\beta'})]^{1+\beta'}_{1+1+\beta'} \quad \text{(IH-12-II)}$$

$$\psi_{1+\beta'} = [F(\phi_{\beta'}, \psi_{\beta'})]^{1+1+\beta'}_{1+\beta'} \quad \text{(IH-12-III)}$$

$$p_{\beta',1+\beta'} = \psi_{\beta'} \quad \text{(IH-12-IV)}$$

$$e_{\beta',1+\beta'} = \phi_{\beta'} \quad \text{(IH-12-V)}$$

$$\forall \gamma_0 \prec \beta'. [F(e_{\gamma_0,\beta'}, p_{\gamma_0,\beta'})]^{1+\beta'}_{1+\gamma_0} = p_{1+\gamma_0,1+\beta'} \quad \text{(IH-12-VI)}$$

If β is a limit ordinal and $\gamma_0 \prec \beta$, we have $[F(e_{\gamma_0,\beta}, p_{\gamma_0,\beta})]^{1+\gamma_0}_{1+\beta} = p_{1+\gamma_0,\beta} \circ \psi_\beta$ (IH-13)

Note how the dependent tuple refers to the previous approximations at previous steps of the induction, therefore it is not so clear that this induction is sound. That is why we first consider the three cases of the induction – the zero ordinal, successor ordinals, and limit ordinals – and only after that describe how we can piece that together into one coherent induction where this is legal.

We quickly comment on all of these properties.

- Properties IH-0, IH-1, and IH-2 define the approximations as well as the embedding-projection pairs between them. The embedding-projection pairs between X_γ and X_β are inverses up to γ (IH-3 and IH-4) and moreover they are functorial (IH-5 and IH-6).
- For the final limit construction we need that X_β is already a bounded solution (up to β) of the domain equation. This is stated by properties IH-7, IH-8, IH-9, and IH-10.
- X_β is truncated at β (IH-11).
- Property IH-12 captures the defining equations of X, e, and p for the successor case; IH-12-VI is a statement which follows inductively from the equations and which we need to carry through for the limit case.
- IH-13 is the statement corresponding to IH-12-VI, but for the limit case.

In the statement, we use a stronger form of isomorphisms which is asymmetric, for instance, $p_{\gamma_0,\gamma_1} \circ e_{\gamma_0,\gamma_1} = id$ should hold instead of an equality at γ_0. While this full equality is required, in our setup $\overset{\gamma_0}{=}$ would also suffice since X_{γ_0} is truncated at γ_0.

5.6 Base Case

First note that most of the properties we need to show are trivial as there are no ordinals below 0. The interesting part is the definition of X_0 itself and the maps ϕ_0, ψ_0.

The straightforward approach would be to set $X_0 \triangleq \mathbb{1}$, $\phi_0\ (x) \triangleq \lfloor \star_{F(\mathbb{1},\mathbb{1})} \rfloor_{1+0}$ and $\phi_0\ (x) \triangleq \star_{\mathbb{1}}$. The problem is that these are not inverses up to 0, as $F(\mathbb{1}, \mathbb{1})$ might have more than one element. One way to fix this would be to make equality at 0 trivial in the definition of OFEs (as is done by Svendsen et al. [2016]). Instead, we essentially apply the functor one more time and use that it is locally contractive, thus "shifting

by one ordinal". This leaves us with the following definitions:

$$\begin{aligned} X'_0 &\triangleq \mathbb{1} \\ X_0 &\triangleq [F(X'_0, X'_0)]_0 \\ \phi'_0 : X'_0 \xrightarrow{\text{ne}} X_0 &\triangleq \lambda\_.\lfloor \star_{F(\mathbb{1},\mathbb{1})} \rfloor_0 \\ \psi'_0 : X_0 \xrightarrow{\text{ne}} X'_0 &\triangleq \lambda\_.\star_{\mathbb{1}} \\ \phi_0 : X_0 \xrightarrow{\text{ne}} [F(X_0, X_0)]_{1+0} &\triangleq [F(\phi'_0, \psi'_0)]^0_{1+0} \\ \psi_0 : [F(X_0, X_0)]_{1+0} \xrightarrow{\text{ne}} X_0 &\triangleq [F(\psi'_0, \phi'_0)]^{1+0}_0 \end{aligned}$$

As ϕ'_0 and ψ'_0 are inverses for all $\gamma \prec 0$ (vacuously) and F is locally contractive, ϕ_0 and ψ_0 are inverses up to 0.

5.7 Successor Case

Suppose that we have a solution up to β, featuring in particular an approximation X_β and maps $\phi_\beta : X_\beta \xrightarrow{\text{ne}} [F(X_\beta, X_\beta)]_{1+\beta}$ and $\psi_\beta : [F(X_\beta, X_\beta)]_{1+\beta} \xrightarrow{\text{ne}} X_\beta$ which are inverses up to β.

We then go on to define:

$$\begin{aligned} X_{1+\beta} &\triangleq [F(X_\beta, X_\beta)]_{1+\beta} \\ \phi_{1+\beta} : X_{1+\beta} \xrightarrow{\text{ne}} [F(X_{1+\beta}, X_{1+\beta})]_{1+1+\beta} &\triangleq [F(\psi_\beta, \phi_\beta)]^{1+\beta}_{1+1+\beta} \\ \psi_{1+\beta} : [F(X_{1+\beta}, X_{1+\beta})]_{1+1+\beta} \xrightarrow{\text{ne}} X_{1+\beta} &\triangleq [F(\phi_\beta, \psi_\beta)]^{1+1+\beta}_{1+\beta} \end{aligned}$$

Compared to the previous outline, we have just added the truncations. Again, as F is locally contractive and ϕ_β, ψ_β are inverses up to β, these are inverses up to $1+\beta$. Specifically, we have

$$\begin{aligned} \phi_{1+\beta} \circ \psi_{1+\beta} &= [F(\psi_\beta, \phi_\beta)]^{1+\beta}_{1+1+\beta} \circ [F(\phi_\beta, \psi_\beta)]^{1+1+\beta}_{1+\beta} \\ &\overset{1+\beta}{=} [F(\phi_\beta \circ \psi_\beta, \phi_\beta \circ \psi_\beta)]^{1+1+\beta}_{1+1+\beta} \\ &\overset{1+\beta}{=} [id]^{1+1+\beta}_{1+1+\beta} \overset{1+1+\beta}{=} id \end{aligned}$$

and

$$\begin{aligned} \psi_{1+\beta} \circ \phi_{1+\beta} &= [F(\phi_\beta, \psi_\beta)]^{1+1+\beta}_{1+\beta} \circ [F(\psi_\beta, \phi_\beta)]^{1+\beta}_{1+1+\beta} \\ &\overset{1+\beta}{=} [F(\psi_\beta \circ \phi_\beta, \psi_\beta \circ \phi_\beta)]^{1+\beta}_{1+\beta} \\ &\overset{1+\beta}{=} [id]^{1+\beta}_{1+\beta} \overset{1+\beta}{=} id. \end{aligned}$$

For the latter property, $\overset{1+\beta}{=}$ is in fact equivalent to full equality as $X_{1+\beta}$ is truncated at $1+\beta$.

We can now already show properties IH-11 and IH-12-I - IH-12-III, which follow directly from the definition.

Next, we tackle the embedding-projection pairs. We assume that we have $e_{\gamma_0,\gamma_1} : X_{\gamma_0} \xrightarrow{\text{ne}} X_{\gamma_1}$ and $p_{\gamma_0,\gamma_1} : X_{\gamma_1} \xrightarrow{\text{ne}} X_{\gamma_0}$ for $\gamma_0 \prec \gamma_1 \preceq \beta$ available by induction and that these satisfy IH-3 and IH-4 (inverses) as well as IH-5 and IH-6 (functoriality). Thus, we just have to add the maps $e_{i,j}, p_{i,j}$ where $j = 1+\beta$.

$$e_{\gamma_0,1+\beta} \triangleq \begin{cases} \phi_\beta & \text{for } \gamma_0 = \beta \\ \phi_\beta \circ e_{\gamma_0,\beta} & \text{for } \gamma_0 \prec \beta \end{cases}$$

As the functions used in both cases are non-expansive, it is clear that e is non-expansive.

$$p_{\gamma_0,1+\beta} \triangleq \begin{cases} \psi_\beta & \text{for } \gamma_0 = \beta \\ p_{\gamma_0,\beta} \circ \psi_\beta & \text{for } \gamma_0 \prec \beta \end{cases}$$

We can now verify properties IH-3-IH-6. First, we prove that $p_{\gamma_0,1+\beta} \circ e_{\gamma_0,1+\beta} = id$ (IH-3). This is by case analysis on $\gamma_0 = \beta$ or $\gamma_0 \prec \beta$. Both cases are straightforward to check as $p_{\gamma_0,\beta} \circ e_{\gamma_0,\beta} = id$ and $\psi_\beta \circ \phi_\beta = id$.

Next is $e_{\gamma_0,1+\beta} \circ p_{\gamma_0,1+\beta} \overset{\gamma_0}{=} id$ (IH-4). This follows again by a case analysis and the inductive hypothesis.

The functoriality facts IH-5 and IH-6 follow similarly from the definition and the inductive hypothesis. It remains to prove the rather technical facts IH-12-iv, IH-12-v, and IH-12-vi, since IH-13 holds vacuously as $1 + \beta$ is not a limit ordinal. IH-12-iv and IH-12-v hold straightforwardly by the defining equations. IH-12-vi is by far the most complicated equation:

$$\forall \gamma_0 \prec \beta. [F(e_{\gamma_0,\beta}, p_{\gamma_0,\beta})]^{1+\beta}_{1+\gamma_0} = p_{1+\gamma_0,1+\beta}.$$

We do a case analysis on $1 + \gamma_0 = \beta$ or $1 + \gamma_0 \prec \beta$, along the definition of $p_{1+\gamma_0,1+\beta}$:

- If $1 + \gamma_0 = \beta$, we prove $[F(e_{\gamma_0,1+\gamma_0}, p_{\gamma_0,1+\gamma_0})]^{1+1+\gamma_0}_{1+\gamma_0} = \psi_{1+\gamma_0}$ by definition of $p_{\_,1+\beta}$. By IH-12-iv and IH-12-v, we can rewrite the LHS to

  $$[F(\phi_{\gamma_0}, \psi_{\gamma_0})]^{1+1+\gamma_0}_{1+\gamma_0} = \psi_{1+\gamma_0}.$$

  With IH-12-iii we are then done.

- If $1 + \gamma_0 \prec \beta$, we show $[F(e_{\gamma_0,\beta}, p_{\gamma_0,\beta})]^{1+\beta}_{1+\gamma_0} = p_{1+\gamma_0,\beta} \circ \psi_\beta$ by definition of $p_{\_,1+\beta}$. Our goal is to apply the inductive hypothesis. For that, we distinguish whether β is a limit ordinal or a successor ordinal $\beta = 1 + \beta'$.

  - In case β is a limit ordinal, the statement that we need to prove is exactly the inductive hypothesis IH-13.
  - If $\beta = 1 + \beta'$, we show

    $$[F(e_{\gamma_0,1+\beta'}, p_{\gamma_0,1+\beta'})]^{1+1+\beta'}_{1+\gamma_0} = p_{1+\gamma_0,1+\beta'} \circ \psi_{1+\beta'}$$

    Due to truncation, it actually suffices to show the equality up to $1 + \gamma_0 \preceq \beta'$.

    $$\begin{aligned} & [F(e_{\gamma_0,1+\beta'}, p_{\gamma_0,1+\beta'})]^{1+1+\beta'}_{1+\gamma_0} && \mid \text{functoriality} \\ \overset{1+\beta'}{=} & [F(e_{\gamma_0,\beta'}, p_{\gamma_0,\beta'})]^{1+\beta'}_{1+\gamma_0} \circ [F(e_{\beta',1+\beta'}, p_{\beta',1+\beta'})]^{1+1+\beta'}_{1+\beta'} && \mid \text{IH-12-iv, IH-12-v, IH-12-vi} \\ = & p_{1+\gamma_0,1+\beta'} \circ [F(\phi_{\beta'}, \psi_{\beta'})]^{1+1+\beta'}_{1+\beta'} && \mid \text{IH-12-iii} \\ = & p_{1+\gamma_0,1+\beta'} \circ \psi_{1+\beta'} \end{aligned}$$

5.8 Limit Case

Finally, we dicuss the limit case. This case is the most critical one. Assume that β is a limit ordinal and we already have approximations for all smaller ordinals $\gamma \prec \beta$; in particular, that we have the embeddings and projections $e_{\gamma_0,\gamma_1}, p_{\gamma_0,\gamma_1}$ for all $\gamma_0 \prec \gamma_1 \prec \beta$.

As in the case for the 0 ordinal, we proceed in two steps by first defining an approximation X'_β which is a solution up to $\prec \beta$ and then applying the functor F once more to obtain the final approximation X_β, using local contractivity.

5.8.1 Step 1: Pre-solution

Essentially, we need to take an inverse limit of all the previous approximations X_γ for $\gamma \prec \beta$, as shown graphically in the outline. We do this in a slight variation, which is however equivalent. Namely, define

$$X'_\beta \triangleq \Sigma x : \prod_{\gamma \prec \beta} [F(X_\gamma, X_\gamma)]_{1+\gamma}. \forall \gamma_0 \prec \gamma_1 \prec \beta. x_{\gamma_0} = [F(e_{\gamma_0,\gamma_1}, p_{\gamma_0,\gamma_1})]^{1+\gamma_1}_{1+\gamma_0}(x_{\gamma_1})$$

as a dependent sum where the predicate "equalises" the components of the product. Hence, for every $x : X'_\beta$, we have

$$\forall \gamma_0 \prec \gamma_1 \prec \beta. x_{\gamma_0} = [F(e_{\gamma_0,\gamma_1}, p_{\gamma_0,\gamma_1})]^{1+\gamma_1}_{1+\gamma_0}(x_{\gamma_1}). \qquad \text{(LIM-EQUALISE)}$$

In the definition, we first apply the functor one more time before taking the limit. We could also directly take the limit and then do this application later when defining the maps ϕ_β and ψ_β. Morally, due to inductive hypotheses IH-12-I and IH-12-VI, $[F(X_\gamma, X_\gamma)]_{1+\gamma}$ is $X_{1+\gamma}$ and $[F(e_{\gamma_0,\gamma_1}, p_{\gamma_0,\gamma_1})]^{1+\gamma_1}_{1+\gamma_0}(x_{\gamma_1})$ is just $p_{1+\gamma_0,1+\gamma_1}$.

We use the notation $x_\gamma \triangleq \pi_\gamma x$ for projecting out a particular component of the limit. Equality on X'_β is defined point-wise. For $x, y : X'_\beta$, $x \overset{\alpha}{=} y$ is equivalent to $x_\gamma \overset{\alpha}{=} y_\gamma$ for all $\gamma \prec \beta$.

Note that X'_β is truncated at β as its components are truncated at some $\gamma \prec \beta$.

We make the crucial observation that X'_β does not seem to be a COFE: if we have a bounded chain of elements $(c_\gamma)_{\gamma \prec \alpha}$ and we take the limit point-wise, then the limits will not necessarily fulfill LIM-EQUALISE again. The critical case is when $\alpha \prec \beta$: then we need to show that

$$\left(\lim_{\gamma \prec \alpha} c_\gamma\right)_{\gamma_0} = [F(e_{\gamma_0,\gamma_1}, p_{\gamma_0,\gamma_1})]^{1+\gamma_1}_{1+\gamma_0}(x_{\gamma_1})\left(\lim_{\gamma \prec \alpha} c_\gamma\right)_{\gamma_1}$$

for $\gamma_0 \prec \gamma_1 \prec \beta$, assuming that this holds for each element of the chain c. By truncation, it suffices to show this equality at γ_0. If $\beta \preceq \alpha$ (implying $\gamma_0 \prec \alpha$), we can just make use of the fact that $\lim_{\gamma \prec \alpha} c_\gamma \overset{\gamma_0}{=} c_{\gamma_0}$, but if $\alpha \preceq \gamma_0$, we cannot reduce this equation to the components of the chain.

It does not seem clear how one could define the limits differently such that the conditions posed by point-wise equality are still met. Hence, it will be important that our functor works on arbitrary OFEs[14].

[14] As mentioned earlier, one can however define limits for chains of length $\succ \beta$ and in this way also solve domain equations for functors which work for such "lower-bounded COFEs". Currently, this is not mechanised in Coq due to dependent typing ugliness. In the case of finite index types ($\mathbb{N}$), lower-bounded COFEs correspond to COFEs, such that our proof subsumes the finite Iris domain equation solver if one assumes classical logic.

Defining the embeddings and projections Now we can define the maps $e'_{\gamma,\beta} : X_\gamma \xrightarrow{\text{ne}} X'_\beta, p'_{\gamma,\beta} : X'_\beta \xrightarrow{\text{ne}} X_\gamma$. $e'_{\gamma,\beta}$ needs to map into the inverse limit, hence we define it component-wise for $\gamma' \prec \beta$:

$$e'_{\gamma,\beta}(x)_{\gamma'} \triangleq \begin{cases} p_{1+\gamma',\gamma} & \text{for } 1+\gamma' \prec \gamma \\ x & \text{for } 1+\gamma' = \gamma \\ e_{\gamma,1+\gamma'} & \text{for } 1+\gamma' \succ \gamma \end{cases}$$

Of course, we need to show that this is well-defined, *i.e.,* non-expansive and the result does actually satisfy LIM-EQUALISE in the definition of X'_β. Non-expansiveness follows by case analysis. The equalisation needs more case analyses, along the definition of $e'_{\gamma,\beta}$. Most cases are contradictory and for the other ones we need equations IH-3-IH-6 from the inductive hypothesis as well as, crucially, IH-13. Defining $p'_{\gamma,\beta}$ is easier. We take $p'_{\gamma,\beta}(x) \triangleq \psi_\gamma(x_\gamma)$.

We now verify properties IH-3-IH-6.

- We have
$$p'_{\gamma,\beta} \circ e'_{\gamma,\beta} = \psi_\gamma \circ e_{\gamma,1+\gamma} = p_{\gamma,1+\gamma} \circ e_{\gamma,1+\gamma} = id$$
where the second step follows by IH-12-IV.
- We use that equality on X'_β is defined point-wise. So, suppose that $\delta \prec \beta$. We show that
$$e'_{\gamma,\beta}(p'_{\gamma,\beta}x)_\delta \overset{\gamma}{=} x_\delta.$$
A case analysis along the definition of $e'_{\gamma,\beta}$ is required. All of the cases follow relatively directly using the properties from the IH, in particular IH-12-IV, IH-12-VI, and IH-3-IH-6, as well as LIM-EQUALISE. For the case that $\gamma \prec 1+\delta$, another case analysis on the relation of γ and δ is helpful.
- We show that $e'_{\gamma_0,\beta} = e'_{\gamma_1,\beta} \circ e_{\gamma_0,\gamma_1}$ for $\gamma_0 \prec \gamma_1 \prec \beta$. We use point-wise equality and make a case analysis on the relation of $1+\gamma, \gamma_0$, and γ_1. All of the cases are straightforward.
- $p'_{\gamma_0,\beta} = p_{\gamma_0,\gamma_1} \circ p'_{\gamma_1,\beta}$ follows directly using equalisation and functoriality.

Defining the bounded isomorphisms Finally, we can turn to the interesting part: defining $\phi'_\beta : X'_\beta \xrightarrow{\text{ne}} [F(X'_\beta, X'_\beta)]_\beta$ and $\psi'_\beta : [F(X'_\beta, X'_\beta)]_\beta \xrightarrow{\text{ne}} X'_\beta$. As we will see below, we could not define these maps between X'_β and $[F(X'_\beta, X'_\beta)]_{1+\beta}$, so the two-stepped definition of X_β (where we first truncate at β') really seems to be necessary.

For ψ'_β, we take

$$\psi'_\beta(x)_\gamma \triangleq [F(e'_{\gamma,\beta}, p'_{\gamma,\beta})]^\beta_{1+\gamma}(x),$$

following the idea of lifting given in the outline. Well-definedness is easy to check.

ϕ'_β is more complicated: we can easily map from each $F(X_\gamma, X_\gamma)$ into $F(X'_\beta, X'_\beta)$, but this needs to be done for all components simultaneously in order to obtain that ψ'_β and ϕ'_β are inverses up to $\prec \beta$. The only possibility here is to take a limit in $[F(X'_\beta, X'_\beta)]_\beta$.

$$\phi'_\beta(x) \triangleq \lim_{\gamma \prec \beta} [F(p'_{\gamma,\beta}, e'_{\gamma,\beta})]^{1+\gamma}_\beta(x_\gamma)$$

Proving that this is well-defined is quite instructive and sheds some light on where we need some properties.

- First of all, we need to prove that this is actually a bounded chain. So let $\gamma' \prec \gamma \prec \beta$ be given.

$$\begin{aligned}
&[F(p'_{\gamma',\beta}, e'_{\gamma',\beta})]_\beta^{1+\gamma'}(x_{\gamma'}) && \mid \textsc{lim-equalise}\\
=&[F(p'_{\gamma',\beta}, e'_{\gamma',\beta})]_\beta^{1+\gamma'}([F(e_{\gamma',\gamma}, p_{\gamma',\gamma})]_{1+\gamma'}^{1+\gamma}(x_\gamma))\\
\overset{1+\gamma'}{=}&[F(e_{\gamma',\gamma} \circ p'_{\gamma',\beta}, e'_{\gamma',\beta} \circ p_{\gamma',\gamma})]_\beta^{1+\gamma}(x_\gamma) && \mid \text{IH-3-IH-6}\\
\overset{\gamma'}{=}&[F(p'_{\gamma,\beta}, p'_{\gamma,\beta})]_\beta^{1+\gamma}(x_\gamma)
\end{aligned}$$

  where the last step requires functoriality and that $e_{\gamma',\gamma}, p_{\gamma',\gamma}$ are inverses up to γ'.

- Now we show that the map is non-expansive. Suppose that $x, y : X'_\beta$ satisfy $x \overset{\alpha}{=} y$. We show

$$\lim_{\gamma\prec\beta}[F(p'_{\gamma,\beta}, e'_{\gamma,\beta})]_\beta^{1+\gamma}(x_\gamma) \overset{\alpha}{=} \lim_{\gamma\prec\beta}[F(p'_{\gamma,\beta}, e'_{\gamma,\beta})]_\beta^{1+\gamma}(y_\gamma).$$

  Do a case analysis on the relation of α and β. If $\alpha \prec \beta$, then it suffices to show

$$[F(p'_{\alpha,\beta}, e'_{\alpha,\beta})]_\beta^{1+\alpha}(x_\alpha) \overset{\alpha}{=} [F(p'_{\alpha,\beta}, e'_{\alpha,\beta})]_\beta^{1+\alpha}(y_\alpha),$$

  which is straightforward as the involved maps are non-expansive.

  In the case that $\alpha \succeq \beta$, we use truncation at β, so that it suffices to show[15]

$$\lim_{\gamma\prec\beta}[F(p'_{\gamma,\beta}, e'_{\gamma,\beta})]_\beta^{1+\gamma}(x_\gamma) \overset{\beta}{=} \lim_{\gamma\prec\beta}[F(p'_{\gamma,\beta}, e'_{\gamma,\beta})]_\beta^{1+\gamma}(y_\gamma).$$

  By applying Limit Uniqueness **(A1')**, we can then reduce this case to the previous one.

Next, we prove properties IH-9 and IH-10. We start with an auxiliary fact needed for IH-9, essentially stating that, for an element of the inverse limit, projecting out the γ'-th component, expanding back to an element of X'_β, and then mapping back down to an element of X_γ is the same (up to γ') as directly projecting to the γ-th component.

Fact 5. *For $\gamma, \gamma' \prec \beta$ and $x : X'_\beta$, we have*

$$[F(p'_{\gamma',\beta} \circ e'_{\gamma,\beta}, p'_{\gamma,\beta} \circ e'_{\gamma',\beta})]_{1+\gamma}^{1+\gamma'}(x_{\gamma'}) \overset{\gamma'}{=} x_\gamma$$

Proof. Case analysis, LIM-EQUALISE, and properties of e, p. □

- For IH-9, we show $\psi'_\beta \circ \phi'_\beta = id$. By point-wise equality and truncation of the γ-th component at $1+\gamma$, we show

$$[F(e'_{\gamma,\beta}, p'_{\gamma,\beta})]_{1+\gamma}^{\beta}(\lim_{\gamma'\prec\beta}[F(p'_{\gamma',\beta}, e'_{\gamma',\beta})]_\beta^{1+\gamma'}(x_{\gamma'})) \overset{1+\gamma}{=} x_\gamma.$$

  Now we use Fact 1 to move the map inside the limit, which we can do as $1+\gamma' \prec \beta$ (since β is a limit ordinal).

  Here is the point where truncating the OFEs explicitly pays off: due to the truncation, we can actually move the map inside the limit. Svendsen et al. [2016] do not truncate their OFEs and despite that

[15] At this point, truncating $F(X'_\beta, X'_\beta)$ at $1+\beta$ would break without strong limit uniqueness.

(mistakenly, as we showed above) use this commutation property. It does not seem clear how just Strongly Unique Limits could help here *without* using them to truncate the OFEs (*i.e.,* in a more "direct" way).

$$
\begin{aligned}
&\lim_{\gamma' \prec \beta} [F(e'_{\gamma,\beta}, p'_{\gamma,\beta})]^{\beta}_{1+\gamma}([F(p'_{\gamma',\beta}, e'_{\gamma',\beta})]^{1+\gamma'}_{\beta}(x_{\gamma'})) && \mid \text{Fact 3, composition} \\
&\overset{<\beta}{=} \lim_{\gamma' \prec \beta} [F(p'_{\gamma',\beta} \circ e'_{\gamma,\beta}, p'_{\gamma,\beta} \circ e'_{\gamma',\beta})]^{1+\gamma'}_{1+\gamma}(x_{\gamma'}) && \mid \text{Facts 3,5} \\
&\overset{<\beta}{=} \lim_{\gamma' \prec \beta} x_{\gamma} && \mid \textsc{cofe-bcompl} \\
&\overset{<\beta}{=} x_{\gamma}
\end{aligned}
$$

All the $\overset{<\beta}{=}$ equalities are in particular $\overset{1+\gamma}{=}$ equalities, since $\gamma \prec \beta$ and (since β is a limit ordinal) therefore also $1 + \gamma \prec \beta$.

- We can only prove a weaker version of property IH-10 for this preliminary definition:

$$\phi'_{\beta} \circ \psi'_{\beta} \overset{<\beta}{=} id$$

Again, we use weak limit uniqueness (Fact 3).

$$
\begin{aligned}
&\lim_{\gamma \prec \beta} [F(p'_{\gamma,\beta}, e'_{\gamma,\beta})]^{1+\gamma}_{\beta}([F(e'_{\gamma,\beta}, p'_{\gamma,\beta})]^{\beta}_{1+\gamma}(x)) && \mid \text{Fact 3, IH-5, IH-6} \\
&\overset{<\beta}{=} \lim_{\gamma \prec \beta} [F(e'_{\gamma,\beta} \circ p'_{\gamma,\beta}, e'_{\gamma,\beta} \circ p'_{\gamma,\beta})]^{\beta}_{\beta}(x) && \mid \text{Fact 3, IH-3, IH-4} \\
&\overset{<\beta}{=} \lim_{\gamma \prec \beta} x && \mid \textsc{cofe-bcompl} \\
&\overset{<\beta}{=} x
\end{aligned}
$$

The last step is what prevents us from proving an equality at β, even though we could use the assumed limit uniqueness to make the previous steps hold at $\overset{\beta}{=}$.

Finally, we show that this preliminary solution satisfies property IH-13:

$$[F(e'_{\gamma,\beta}, p'_{\gamma,\beta})]^{\beta}_{1+\gamma} = p'_{1+\gamma,\beta} \circ \psi'_{\beta}$$

The proof is mostly straightforward and uses truncation, IH-10 which we just proved, LIM-EQUALISE, and some of the inductive hypotheses.

5.8.2 Step 2: Full Approximation

In the previous subsection, we have defined an inverse limit which is already close to satisfying the requirements posed by the inductive proof, but falls short in three places: first of all, the maps go into $[F(X'_{\beta}, X'_{\beta})]_{\beta}$, truncated at β (not at $\beta + 1$); secondly, ϕ'_{β} and ψ'_{β} are only inverses up to $\prec \beta$; finally, X'_{β} isn't even a COFE.

We fix this in a similar way as for the base case, essentially by applying the functor F one more time.

$$\begin{aligned} X_\beta &\triangleq [F(X'_\beta, X'_\beta)]_\beta \\ \phi_\beta : X_\beta \xrightarrow{\text{ne}} [F(X_\beta, X_\beta)]_{1+\beta} &\triangleq [F(\psi_\beta, \phi_\beta)]^\beta_{1+\beta} \\ \psi_\beta : [F(X_\beta, X_\beta)]_{1+\beta} \xrightarrow{\text{ne}} X_\beta &\triangleq [F(\phi_\beta, \psi_\beta)]^{1+\beta}_\beta \end{aligned}$$

One can directly see that now, ϕ_β and ψ_β are inverses up to β as F is locally contractive. Next, we lift the embedding-projection maps:

$$\begin{aligned} e_{\gamma,\beta} : X_\gamma \xrightarrow{\text{ne}} X_\beta &\triangleq \phi'_\beta \circ e'_{\gamma,\beta} \\ p_{\gamma,\beta} : X_\beta \xrightarrow{\text{ne}} X_\gamma &\triangleq p'_{\gamma,\beta} \circ \psi'_\beta \end{aligned}$$

These are trivially inverses and functorial as required. To close the proof, property IH-13 needs to be shown:

$$[F(e_{\gamma,\beta}, p_{\gamma,\beta})]^{1+\beta}_{1+\gamma} = p_{1+\gamma,\beta} \circ \psi_\beta$$

This follows from the preliminary version of IH-13 which we showed in the previous subsection.

Property IH-12 only applies to the successor case and thus need not be shown. Truncation IH-11 holds by our definition of X_β.

5.9 Deriving the Final Isomorphism

Now, assume that we have an approximation for every ordinal β available, as well as proofs that these are bounded solutions of F up to β (witnessed by ϕ_β, ψ_β), and additionally embeddings/projections between all of them.

We now derive the full solution and isomorphism by taking a limit of these approximations. The construction is very similar to the limit case.

$$X \triangleq \Sigma x : \prod_\gamma [F(X_\gamma, X_\gamma)]_{1+\gamma}. \forall \gamma_0 \prec \gamma_1 . x_{\gamma_0} = [F(e_{\gamma_0,\gamma_1}, p_{\gamma_0,\gamma_1})]^{1+\gamma_1}_{1+\gamma_0}(x_{\gamma_1})$$

Define embeddings and projections as follows:

$$\begin{aligned} e_\gamma : X_\gamma \xrightarrow{\text{ne}} X & \\ e_\gamma(x)_{\gamma'} \triangleq \begin{cases} p_{1+\gamma',\gamma} & \text{for } 1+\gamma' \prec \gamma \\ x & \text{for } 1+\gamma' = \gamma \\ e_{\gamma,1+\gamma'} & \text{for } 1+\gamma' \succ \gamma \end{cases} & \\ p_\gamma : X \xrightarrow{\text{ne}} X_\gamma \triangleq \lambda x . \psi_\gamma(x_\gamma) & \end{aligned}$$

We can, similarly to the limit case, show that these are functorial and bounded inverses.

More interesting is the definition of the maps $\phi_X : X \xrightarrow{\text{ne}} F(X,X)$ and $\psi_X : F(X,X) \xrightarrow{\text{ne}} X$ which is slightly different as we take an unbounded limit and do not have the truncation.

$$\begin{aligned} \phi_X(x) &\triangleq \lim_\gamma F(p_\gamma, e_\gamma)\lceil x_\gamma \rceil^{1+\gamma} \\ \psi_X(x)_\gamma &\triangleq \lfloor F(e_\gamma, p_\gamma)(x) \rfloor_{1+\gamma} \end{aligned}$$

We verify that these form an isomorphism.

$$\begin{aligned}
\phi_X(\psi_X(x)) &= \lim_\gamma F(p_\gamma, e_\gamma)\lceil\lfloor F(e_\gamma, p_\gamma)(x)\rfloor_{1+\gamma}\rceil^{1+\gamma} && \mid \text{Fact 4}\\
&= \lim_\gamma F(e_\gamma \circ p_\gamma, e_\gamma \circ p_\gamma)(x) && \mid \text{Fact 4, IH-3, IH-4}\\
&= \lim_\gamma x && \mid \textsc{cofe-compl}\\
&= x,
\end{aligned}$$

using Fact 4 to rewrite inside the limit.

$$\begin{aligned}
\psi_X(\phi_X(x))_{\gamma'} &= \lfloor F(e'_\gamma, p'_\gamma)(\lim_\gamma F(p_\gamma, e_\gamma)(\lceil x_\gamma\rceil^{1+\gamma}))\rfloor_{1+\gamma'} && \mid \text{Fact 2}\\
&= \lim_\gamma \lfloor F(e'_\gamma, p'_\gamma)(F(p_\gamma, e_\gamma)(\lceil x_\gamma\rceil^{1+\gamma}))\rfloor_{1+\gamma'} && \mid \textsc{cofe-compl}\\
&= \lfloor F(e_{\gamma'}, p_{\gamma'})(F(p_{1+\gamma'}, e_{1+\gamma'})(\lceil x_{1+\gamma'}\rceil^{1+1+\gamma'}))\rfloor_{1+\gamma'}
\end{aligned}$$

In this last step we have equated the limit to its $1+\gamma'$-th component and used truncation at $1+\gamma'$ to get full equality. Using functoriality, we get

$$\begin{aligned}
&\lfloor F(e_{\gamma'}, p_{\gamma'})(F(p_{1+\gamma'}, e_{1+\gamma'})(\lceil x_{1+\gamma'}\rceil^{1+1+\gamma'}))\rfloor_{1+\gamma'}\\
=&\lfloor F(e_{\gamma'}, p_{\gamma'})(F(p_{\gamma',1+\gamma'} \circ p_{\gamma'}, e_{\gamma'} \circ e_{\gamma',1+\gamma'})(\lceil x_{1+\gamma'}\rceil^{1+1+\gamma'}))\rfloor_{1+\gamma'} && \mid \text{functoriality}\\
=&\lfloor (F(p_{\gamma',1+\gamma'} \circ p_{\gamma'} \circ e_{\gamma'}, p_{\gamma'} \circ e_{\gamma'} \circ e_{\gamma',1+\gamma'})(\lceil x_{1+\gamma'}\rceil^{1+1+\gamma'}))\rfloor_{1+\gamma'} && \mid \text{IH-3, IH-4}\\
=&\lfloor F(p_{\gamma',1+\gamma'}, e_{\gamma',1+\gamma'})(\lceil x_{1+\gamma'}\rceil^{1+1+\gamma'})\rfloor_{1+\gamma'} && \mid \textsc{lim-equalise}\\
=&x_{\gamma'}
\end{aligned}$$

In the second step, we have used that F is locally contractive, so that we get an equality at $1+\gamma'$ from $p_{\gamma'}, e_{\gamma'}$ being inverses up to γ'. The last step holds by the equalisation property LIM-EQUALISE of the inverse limit X.

To conclude the proof, we have to show that X is a COFE. A priori, this again need not hold as **COFE** does not, in general, have limits. But as $F(X,X)$ is a COFE and $X \simeq F(X,X)$, we can just equip X with limits by mapping chains to $F(X,X)$, taking the limit there, and mapping back.

5.10 Closing the Induction

Up to now, we have quietly assumed that we can actually define an approximation X_β and the maps $e_{\gamma,\beta}, p_{\gamma,\beta}$ at the same time, having access to the X_γ we defined at previous steps. However, a "plain" transfinite induction does not allow this: in the statement $P : \mathcal{I} \to \mathbb{T}$ itself, we would need access to the inductive hypothesis (*i.e.,* the smaller X_γ) to say anything about the projections $e_{\gamma,\beta}, p_{\gamma,\beta}$.

To solve this problem, we have to define *a full chain* of *all* approximations up to β at step β. In this way, the statement $P : \mathcal{I} \to \mathbb{T}$ we want to derive for all indices is well-defined as it need not refer to the inductive hypothesis.

For various reasons which will become clear as we present the construction, we do not simply parameterise the statement over an ordinal, but over a predicate on ordinals: $\mathcal{A} : (\mathcal{I} \to \mathit{Prop}) \to \mathbb{T}$. Intuitively, we can then obtain the above predicate $P\ \beta$ by instantiating $\mathcal{A}(\lambda\gamma.\gamma \preceq \beta)$. Formally, $\mathcal{A}\ p$, where $p : \mathcal{I} \to \mathit{Prop}$,

has the following important components, where the proof components are analogous to the ones above and therefore omitted:

$$\begin{aligned}
\forall\gamma.p\ \gamma \to X_\gamma &: \text{COFE} \\
\forall\gamma_0 \prec \gamma_1.e_{\gamma_0,\gamma_1} &: p\ \gamma_0 \to p\ \gamma_1 \to X_{\gamma_0} \xrightarrow{\text{ne}} X_{\gamma_1} \\
\forall\gamma_0 \prec \gamma_1.p_{\gamma_0,\gamma_1} &: p\ \gamma_0 \to p\ \gamma_1 \to X_{\gamma_1} \xrightarrow{\text{ne}} X_{\gamma_0} \\
\forall\gamma.\phi_\gamma &: p\ \gamma \to X_\gamma \xrightarrow{\text{ne}} [F(X_\gamma, X_\gamma)]_{1+\gamma} \\
\forall\gamma.\psi_\gamma &: p\ \gamma \to [F(X_\gamma, X_\gamma)]_{1+\gamma} \xrightarrow{\text{ne}} X_\gamma
\end{aligned}$$

In this setting, we can just regard the list of components given initially, which refer to previous approximations, as an extension $\mathcal{E} : \forall\beta.\mathcal{A}(\lambda\gamma.\gamma \prec \beta) \to \mathbb{T}$ which is parameterised over a collection of approximations for all smaller ordinals. As we have seen above, in each step of the induction, we define such an extension to the previous approximations.

We can now look formally at the structure of the "induction" given above:

- in the base case, we define $A_0 : \mathcal{A}(\lambda\gamma.\gamma \preceq 0)$.
- in the successor case, we assume $IH : \mathcal{A}(\lambda\gamma.\gamma \prec 1 + \beta)$ and define $e_{1+\beta} : \mathcal{E}\ (1 + \beta)\ IH$,
- in the limit case, we assume $IH : \mathcal{A}(\lambda\gamma.\gamma \prec \beta)$ and define $e_\beta : \mathcal{E}\ \beta\ IH$,
- for the final limit, we assume $IH : \mathcal{A}(\lambda\gamma.\top)$.

With this, we can now clearly see what is missing to close the induction:

- for the successor and limit cases, we need to *extend* the inductive hypothesis $IH : \mathcal{A}\ (\lambda\gamma, \gamma \prec \beta')$ by the extension $e : \mathcal{E}\ \beta'\ IH$ into a new collection of approximations extend $IH\ e : \mathcal{A}(\lambda\gamma.\gamma \preceq \beta')$.
- for the limit case, the inductive hypothesis we assume is not what we get from a transfinite induction; actually, our inductive hypothesis would be $IH' : \forall\gamma \prec \beta.\mathcal{A}(\lambda\gamma'.\gamma' \preceq \gamma)$. We need to *merge* all these approximations into one coherent one.
- after completing the full transfinite induction, we end up with the statement $\forall\beta.\mathcal{A}(\lambda\gamma.\gamma \preceq \beta)$, but we would like to obtain $\mathcal{A}(\lambda\gamma.\top)$ to apply the final limit construction. We need a similar merging operation as for the limit case.

The extension operation $\text{extend} : \forall\gamma(A : \mathcal{A}\ (\lambda\gamma'.\gamma' \prec \gamma)).\mathcal{E}\ A \to \mathcal{A}\ (\lambda\gamma'.\gamma' \preceq \gamma)$ essentially just straps the additional approximation onto the sequence of approximations we already have. With this, we get the following inductive steps:

$$\begin{aligned}
A_0 &: \mathcal{A}(\lambda\gamma.\gamma \preceq 0) \\
A_{\mathbf{S}} &: \forall\beta.\mathcal{A}(\lambda\gamma.\gamma \prec 1 + \beta) \to \mathcal{A}(\lambda\gamma.\gamma \preceq 1 + \beta) \\
A_{\mathbf{lim}} &: \forall\beta.\mathcal{A}(\lambda\gamma.\gamma \prec \beta) \to \mathcal{A}(\lambda\gamma.\gamma \preceq \beta) \qquad \text{for a limit ordinal } \beta
\end{aligned}$$

Clearly, for the merging operation, we cannot just merge arbitrary approximations, but they need to agree in some way. In the induction, we need to carry this agreement through. The idea here is that two approximations $a_0 : \mathcal{A}(\lambda\gamma.\gamma \preceq \beta_0)$ and $a_1 : \mathcal{A}(\lambda\gamma.\gamma \preceq \beta_1)$ *must have been defined in the same way* for all

$\gamma \preceq \min(\beta_0, \beta_1)$, as they originate from the same induction. Let this form of agreement be captured by a predicate agree : $\forall P_0\ P_1.\mathcal{A}\ P_0 \to \mathcal{A}\ P_1 \to \mathit{Prop}$[16].

The merging operation

$$\mathsf{merge} : \forall P(IH : \forall \gamma.P\ \gamma \to \mathcal{A}\ (\lambda\gamma'.\gamma' \preceq \gamma)) \to (\forall \gamma_0\ \gamma_1\ H_0\ H_1.\mathsf{agree}\ (IH\ \gamma_0\ H_0)\ (IH\ \gamma_1\ H_1)) \to \mathcal{A}\ P$$

takes, for every γ with $P\ \gamma$, the approximation X_γ of $IH\ \gamma$ to define the new merged collection of approximations. The maps e, p, ϕ, ψ can then be defined canonically since all of the collections agree. We can prove that merge preserves agreement, *i.e.,* two pointwise agreeing chains are merged to two agreeing collections of approximations, and the merged collection itself agrees with each of the collections of the chain from which it was created.

We need a similar form of agreement for extensions:

$$\mathsf{eagree} : \forall P_0\ P_1(A_0 : \mathcal{A}\ P_0)(A_1 : \mathcal{A}\ P_1).\mathsf{agree}\ A_0\ A_1 \to \mathit{Prop}.$$

Essentially, we prove that the successor and limit case constructions preserve agreement, in the sense that, if we put agreeing collections in, we get agreeing extensions out. They are coherent in a similar way as the merging operation: they preserves agreement and the resulting collection of approximations agrees with the collection we put in.

The only thing which is missing now is to prove that we can actually apply the merging operation when we need it, that is, we need to prove that the collections we get by the inductive hypothesis agree. We define a specification which exactly captures the structure of the induction:

$$\begin{aligned}
\mathsf{spec} &: \forall \gamma.\mathcal{A}\ (\lambda\gamma'.\gamma' \preceq \gamma) \to \mathit{Prop} \\
\mathsf{spec}_0 &: \mathsf{spec} A_0 \\
\mathsf{spec}_{\mathbf{S}} &: \forall \beta\ A.\mathsf{spec}\ \beta\ A \to \mathsf{spec}(1+\beta)(A_{\mathbf{S}}\ A) \\
\mathsf{spec}_{\mathbf{lim}} &: \forall \beta\ (IH : \forall \gamma.\gamma \prec \beta \to \mathcal{A}\ (\lambda\gamma'.\gamma' \preceq \gamma)), \\
&\qquad (\forall \gamma\ H_\gamma.\mathsf{spec}(IH\gamma H_\gamma)) \\
&\qquad \to (\forall \gamma_0\ \gamma_1\ H_0\ H_1.\mathsf{agree}(IH\ \gamma_0\ H_0)(IH\ \gamma_1\ H_1)) \\
&\qquad \to \mathsf{spec}\ \beta\ (A_{\mathbf{lim}}\ IH\ (\mathsf{merge}\ IH))
\end{aligned}$$

By transfinite induction we can prove:

Fact 6. *If spec A_0 and spec A_1, then agree $A_0\ A_1$.*

Now, by transfinite recursion we can define for every index β a dependent pair of type $\Sigma A : \mathcal{A}(\lambda\gamma.\gamma \preceq \beta).\mathsf{spec} A$, using the inductive steps we considered before. In the limit case, we use the previous fact to prove agreement for all approximations of the inductive hypothesis in order to apply the merging operation. By the previous fact, all of these approximations do then agree; thus, we can apply the merge operation to obtain an approximation $\mathcal{A}(\lambda\gamma.\top)$ as desired.

In the Coq formalisation, this last step to close the induction is setup slightly differently and the specific transfinite induction-recursion principle is derived from a more general well-founded principle.

We remark that, while the uniqueness-based approach for closing the induction is simple and the proofs are rather trivial in an extensional type theory, this construction is extremely technical (though

[16] In our extensional type theory, this just means that the common components (the components where both P_0 and P_1 hold) are equal. In intensional type theories (like Coq's), this says that the OFEs are equal and the maps e, p, ϕ, ψ and bounded limits agree modulo typecasts.

mathematically boring) and rather complicated to mechanise in an intensional type theory such as Coq's due to the involved equalities between OFEs (types).

This completes our description of the recursive domain equation solver.

6 Derived Notions of the Base Logic

In this section we consider rules and definitions that were derived from the base logic in finite Iris.

As it turns out, we can retain most of them in Transfinite Iris, although some rules cannot be derived in the presence of transfinite step-indexing. Luckily, these rules do however still hold in the model, but for proving them we need to break Iris' base logic abstractions. Since some of these rules rely on quite complex derived definitions, we have not listed them as primitives of the base logic in Section 4. We leave it to future work to find suitable abstractions for these notions to restore a proper separation into a base logic.

6.1 Derived Rules about Base Connectives

We collect here some important and frequently used derived proof rules.

Löb

$$(\triangleright P \Rid P) \vdash P \qquad P \Rid Q \vdash P \mathrel{-\!\!*} Q \qquad P * \exists x.\, Q \dashv\vdash \exists x.\, P * Q \qquad P * \forall x.\, Q \vdash \forall x.\, P * Q$$

$$\Box(P * Q) \dashv\vdash \Box P * \Box Q \qquad \Box(P \Rid Q) \vdash \Box P \Rid \Box Q \qquad \Box(P \mathrel{-\!\!*} Q) \vdash \Box P \mathrel{-\!\!*} \Box Q$$

$$\Box(P \mathrel{-\!\!*} Q) \dashv\vdash \Box(P \Rid Q) \qquad \triangleright(P \Rid Q) \vdash \triangleright P \Rid \triangleright Q \qquad \triangleright(P \mathrel{-\!\!*} Q) \vdash \triangleright P \mathrel{-\!\!*} \triangleright Q \qquad \mathsf{True} \vdash \blacksquare \mathsf{True}$$

Noteworthy here is the fact that Löb induction can be derived from $\triangleright$-introduction and the fact that we can take fixed-points of functions where the recursive occurrences are below $\triangleright$ [Löb, 1955].[17] Furthermore, $\mathsf{True} \vdash \blacksquare \mathsf{True}$ can be derived via $\blacksquare$ commuting with universal quantification ranging over the empty type 0. To derive that existential quantifiers commute with separating conjunction requires an intermediate step using a magic wand: From $P * \exists x.Q \vdash \exists x.\, P * Q$ we can deduce $\exists x.\, Q \vdash P \mathrel{-\!\!*} \exists x.\, P * Q$ and then proceed via $\exists$-elimination.

6.2 Persistent Propositions

We call a proposition P *persistent* if $P \vdash \Box P$. These are propositions that "do not own anything", so we can (and will) treat them like "normal" intuitionistic propositions.

Of course, $\Box P$ is persistent for any P. Furthermore, True, False, $t = t'$ as well as $\boxed{a}^{\gamma}$ and $\mathcal{V}(a)$ are persistent. Persistence is preserved by conjunction, disjunction, separating conjunction as well as universal and existential quantification and $\triangleright$.

6.3 Timeless Propositions and Except-0

One of the troubles of working in a step-indexed logic is the "later" modality $\triangleright$. It turns out that we can somewhat mitigate this trouble by working below the following *except-0* modality:

$$\diamond P \triangleq \triangleright \mathsf{False} \vee P$$

Except-0 satisfies the usual laws of a "monadic" modality (similar to, *e.g.,* the update modalities):

EX0-MONO
$$\frac{P \vdash Q}{\diamond P \vdash \diamond Q}$$

EX0-INTRO
$$P \vdash \diamond P$$

EX0-IDEM
$$\diamond\diamond P \vdash \diamond P$$

$$\diamond(P * Q) \dashv\vdash \diamond P * \diamond Q$$
$$\diamond(P \wedge Q) \dashv\vdash \diamond P \wedge \diamond Q$$
$$\diamond(P \vee Q) \dashv\vdash \diamond P \vee \diamond Q$$

$$\diamond\forall x.\, P \dashv\vdash \forall x.\, \diamond P$$
$$\diamond\exists x.\, P \dashv\vdash \exists x.\, \diamond P$$[18]
$$\diamond\Box P \dashv\vdash \Box\diamond P$$
$$\diamond\triangleright P \vdash \triangleright P$$

[17] Also see https://en.wikipedia.org/wiki/L%C3%B6b%27s_theorem.

In particular, from EX0-MONO and EX0-IDEM we can derive a "bind"-like elimination rule:

$$\textsc{ex0-elim}\ \frac{P \vdash \diamond Q}{\diamond P \vdash \diamond Q}$$

This modality is useful because there is a class of propositions which we call *timeless* propositions, for which we have

$$\mathsf{timeless}(P) \triangleq \triangleright P \vdash \diamond P$$

In other words, when working below the except-0 modality, we can *strip away* the later from timeless propositions (using EX0-ELIM):

$$\textsc{ex0-timeless-strip}\ \frac{\mathsf{timeless}(P) \qquad P \vdash \diamond Q}{\triangleright P \vdash \diamond Q}$$

In fact, it turns out that we can strip away later from timeless propositions even when working under the later modality:

$$\textsc{later-timeless-strip}\ \frac{\mathsf{timeless}(P) \qquad P \vdash \triangleright Q}{\triangleright P \vdash \triangleright Q}$$

This follows from $\triangleright P \vdash \triangleright \mathsf{False} \lor P$, and then by straightforward disjunction elimination.

The following rules identify the class of timeless propositions:

$$\frac{\Gamma \vdash \mathsf{timeless}(P) \qquad \Gamma \vdash \mathsf{timeless}(Q)}{\Gamma \vdash \mathsf{timeless}(P \land Q)} \qquad \frac{\Gamma \vdash \mathsf{timeless}(P) \qquad \Gamma \vdash \mathsf{timeless}(Q)}{\Gamma \vdash \mathsf{timeless}(P \lor Q)}$$

$$\frac{\Gamma \vdash \mathsf{timeless}(P) \qquad \Gamma \vdash \mathsf{timeless}(Q)}{\Gamma \vdash \mathsf{timeless}(P * Q)} \qquad \frac{\Gamma \vdash \mathsf{timeless}(P)}{\Gamma \vdash \mathsf{timeless}(\Box P)} \qquad \frac{\Gamma \vdash \mathsf{timeless}(Q)}{\Gamma \vdash \mathsf{timeless}(P \Rightarrow Q)}$$

$$\frac{\Gamma \vdash \mathsf{timeless}(Q)}{\Gamma \vdash \mathsf{timeless}(P \mathrel{-\!\!*} Q)} \qquad \frac{\Gamma, x : \tau \vdash \mathsf{timeless}(P)}{\Gamma \vdash \mathsf{timeless}(\forall x : \tau.\ P)} \qquad \frac{\Gamma, x : \tau \vdash \mathsf{timeless}(P)}{\Gamma \vdash \mathsf{timeless}(\exists x : \tau.\ P)} \qquad \mathsf{timeless}(\mathsf{True})$$

$$\mathsf{timeless}(\mathsf{False}) \qquad \frac{a \text{ is a discrete OFE element}}{\mathsf{timeless}(\mathsf{Own}\ (a))} \qquad \frac{t \text{ or } t' \text{ is a discrete OFE element}}{\mathsf{timeless}(t =_\tau t')}$$

$$\frac{a \text{ is an element of a discrete camera}}{\mathsf{timeless}(\mathcal{V}(a))}$$

The rules highlighted in blue are currently proved in the model as they cannot just be derived anymore, where the last two rules essentially depend on the fact that pure propositions (reflected from the metalogic) are timeless, which we currently have to prove in the model, and the rule about ownership needs to be derived in the model due to the loss of the commuting rule of later with ownership (see §4.3). We leave it to future work to find a suitable (minimal) set of rules that can be incorporated into the base logic to make these laws derived again.

[18] The direction from left to right requires that the type over which the existential quantifier ranges is inhabited.

6.4 Dynamic Composeable Higher-Order Resources

The base logic described in §4 works over an arbitrary camera M defining the structure of the resources. It turns out that we can generalize this further and permit picking cameras "$\Sigma(\mathsf{iProp})$" that depend on the structure of propositions themselves. Of course, iProp is just the syntactic type of propositions; for this to make sense we have to look at the semantics.

Furthermore, there is a composability problem with the given logic: if we have one proof performed with camera M_1, and another proof carried out with a *different* camera M_2, then the two proofs are actually carried out in two *entirely separate logics* and hence cannot be combined.

Finally, in many cases just having a single "instance" of a camera available for reasoning is not enough. For example, when reasoning about a dynamically allocated data structure, every time a new instance of that data structure is created, we will want a fresh resource governing the state of this particular instance. While it would be possible to handle this problem whenever it comes up, it turns out to be useful to provide a general solution.

The purpose of this section is to describe how we solve these issues.

Picking the resources. The key ingredient that we will employ on top of the base logic is to give some more fixed structure to the resources. To instantiate the logic with dynamic higher-order ghost state, the user picks a family of locally contractive bifunctors $(\Sigma_i : \mathbf{OFE}^{\mathrm{op}} \times \mathbf{OFE} \to \mathbf{Camera})_{i \in \mathcal{I}}$. (This is in contrast to the base logic, where the user picks a single, fixed camera that has a unit.)

From this, we construct the bifunctor defining the overall resources as follows:

$$\begin{aligned} \mathit{GName} &\triangleq \mathbb{N} \\ \mathit{ResF}(T^{\mathrm{op}}, T) &\triangleq \prod_{i \in \mathcal{I}} \mathit{GName} \xrightarrow{\mathrm{fin}} \Sigma_i(T^{\mathrm{op}}, T) \end{aligned}$$

We will motivate both the use of a product and the finite partial function below. $\mathit{ResF}(T^{\mathrm{op}}, T)$ is a camera by lifting the individual cameras pointwise, and it has a unit (using the empty finite partial function). Furthermore, since the Σ_i are locally contractive, so is *ResF*.

Now we can write down the recursive domain equation:

$$\mathit{iPreProp} \cong \mathit{UPred}(\mathit{ResF}(\mathit{iPreProp}, \mathit{iPreProp}))$$

Here, *iPreProp* is a COFE defined as the fixed-point of a locally contractive bifunctor, which exists by Theorem 2, so we obtain some object *iPreProp* such that:

$$\begin{aligned} \mathit{Res} &\triangleq \mathit{ResF}(\mathit{iPreProp}, \mathit{iPreProp}) \\ \mathit{iProp} &\triangleq \mathit{UPred}(\mathit{Res}) \\ \xi &: \mathit{iProp} \xrightarrow{\mathrm{ne}} \mathit{iPreProp} \\ \xi^{-1} &: \mathit{iPreProp} \xrightarrow{\mathrm{ne}} \mathit{iProp} \\ \xi(\xi^{-1}(x)) &\triangleq x \\ \xi^{-1}(\xi(x)) &\triangleq x \end{aligned}$$

Now we can instantiate the base logic described in §4 with *Res* as the chosen camera. Effectively, we just defined a way to instantiate the base logic with *Res* as the camera of resources, while providing a way for *Res* to depend on *iPreProp*, which is isomorphic to *iProp*.

We thus obtain all the rules of §4, and furthermore, we can use the maps ξ and ξ^{-1} *in the logic* to convert between logical propositions *iProp* and the domain *iPreProp* which is used in the construction of *Res* – so from elements of *iPreProp*, we can construct the elements that can be owned in our logic.

Proof composability. To make our proofs composeable, we *generalize* our proofs over the family of functors. This is possible because we made *Res* a *product* of all the cameras picked by the user, and because we can actually work with that product "pointwise". So instead of picking a *concrete* family, proofs will assume to be given an *arbitrary* family of functors, plus a proof that this family *contains the functors they need*. Composing two proofs is then merely a matter of conjoining the assumptions they make about the functors. Since the logic is entirely parametric in the choice of functors, there is no trouble reasoning without full knowledge of the family of functors.

Only when the top-level proof is completed we will "close" the proof by picking a concrete family that contains exactly those functors the proof needs.

Dynamic resources. Finally, the use of finite partial functions lets us have as many instances of any camera as we could wish for: Because there can only ever be finitely many instances already allocated, it is always possible to create a fresh instance with any desired (valid) starting state. This is best demonstrated by giving some proof rules.

So let us first define the notion of ghost ownership that we use in this logic. Assuming that the family of functors contains the functor Σ_i at index i, and furthermore assuming that $M_i = \Sigma_i(\textit{iPreProp}, \textit{iPreProp})$, given some $a \in M_i$ we define:

$$\boxed{a : M_i}^{\gamma} \triangleq \mathsf{Own}\left((\ldots, \emptyset, i : [\gamma \leftarrow a], \emptyset, \ldots)\right)$$

This is ownership of the pair (element of the product over all the functors) that has the empty finite partial function in all components *except for* the component corresponding to index i, where we own the element a at index γ in the finite partial function.

We can show the following properties for this form of ownership:

RES-ALLOC
$$\frac{G \text{ infinite} \qquad a \in \mathcal{V}_{M_i}}{\mathsf{True} \vdash \dot{\Rightarrow}\exists \gamma \in G.\, \boxed{a : M_i}^{\gamma}}$$

RES-UPDATE
$$\frac{a \rightsquigarrow_{M_i} B}{\boxed{a : M_i}^{\gamma} \vdash \dot{\Rightarrow}\exists b \in B.\, \boxed{b : M_i}^{\gamma}}$$

RES-EMPTY
$$\frac{\varepsilon \text{ is a unit of } M_i}{\mathsf{True} \vdash \dot{\Rightarrow}\boxed{\varepsilon}^{\gamma}}$$

RES-OP
$$\boxed{a : M_i}^{\gamma} * \boxed{b : M_i}^{\gamma} \dashv\vdash \boxed{a \cdot b : M_i}^{\gamma}$$

RES-VALID
$$\boxed{a : M_i}^{\gamma} \Rightarrow \mathcal{V}_{M_i}(a)$$

RES-TIMELESS
$$\frac{a \text{ is a discrete OFE element}}{\mathsf{timeless}(\boxed{a : M_i}^{\gamma})}$$

Below, we will always work within (an instance of) the logic as described here. Whenever a camera is used in a proof, we implicitly assume it to be available in the global family of functors. We will typically leave the M_i implicit when asserting ghost ownership, as the type of a will be clear from the context.

7 Invariants and Update Modalities

The encoding of invariants and the definition of the *fancy update modality* in Transfinite Iris is unchanged from Iris. We state the encoding here for completeness.

However, in order to alleviate the loss of the later commuting rules for separating conjunction and existentials, which are frequently used when opening invariants, we introduce a new update modality, *logical steps*, which will be used in the definition of our transfinite weakest precondition for safety.

7.1 World Satisfaction and Invariants

To introduce invariants into our logic, we will define weakest precondition to explicitly thread through the proof that all the invariants are maintained throughout program execution. However, in order to be able to access invariants, we will also have to provide a way to *temporarily disable* (or "open") them. To this end, we use tokens that manage which invariants are currently enabled.

We assume to have the following four cameras available:

$$\begin{aligned} \mathit{InvName} &\triangleq \mathbb{N} \\ \textsc{Inv} &\triangleq \textsc{Auth}(\mathit{InvName} \xrightarrow{\text{fin}} \textsc{Ag}(\blacktriangleright \mathit{iPreProp})) \\ \textsc{En} &\triangleq \wp(\mathit{InvName}) \\ \textsc{Dis} &\triangleq \wp^{\text{fin}}(\mathit{InvName}) \end{aligned}$$

The last two are the tokens used for managing invariants, Inv is the monoid used to manage the invariants themselves.

We assume that at the beginning of the verification, instances named $\gamma_{\textsc{State}}$, $\gamma_{\textsc{Inv}}$, $\gamma_{\textsc{En}}$ and $\gamma_{\textsc{Dis}}$ of these cameras have been created, such that these names are globally known.

World Satisfaction. We can now define the proposition W (*world satisfaction*) which ensures that the enabled invariants are actually maintained:

$$W \triangleq \exists I : \mathit{InvName} \xrightarrow{\text{fin}} \mathsf{iProp}.\ \boxed{\bullet \left[\iota \leftarrow \mathsf{ag}(\mathsf{next}(\xi(I(\iota)))) \mid \iota \in \mathrm{dom}(I)\right]}^{\gamma_{\textsc{Inv}}} * \\ \ast_{\iota \in \mathrm{dom}(I)} \left(\triangleright I(\iota) * \boxed{\{\iota\}}^{\gamma_{\textsc{Dis}}} \vee \boxed{\{\iota\}}^{\gamma_{\textsc{En}}} \right)$$

Invariants. The following proposition states that an invariant with name ι exists and maintains proposition P:

$$\boxed{P}^{\iota} \triangleq \boxed{\circ \left[\iota \leftarrow \mathsf{ag}(\mathsf{next}(\xi(P)))\right]}^{\gamma_{\textsc{Inv}}}$$

7.2 Fancy Updates

Next, we define *fancy updates*, which are essentially the same as the basic updates of the base logic (§4), except that they also have access to world satisfaction and can enable and disable invariants:

$${}^{\mathcal{E}_1}\!\Rrightarrow^{\mathcal{E}_2} P \triangleq W * \boxed{\mathcal{E}_1}^{\gamma_{\textsc{En}}} \mathrel{-\!\!*} \dot{\Rrightarrow} \diamond (W * \boxed{\mathcal{E}_2}^{\gamma_{\textsc{En}}} * P)$$

Here, $\mathcal{E}_1$ and $\mathcal{E}_2$ are the *masks* of the view update, defining which invariants have to be (at least!) available before and after the update. We use $\top$ as symbol for the largest possible mask, $\mathbb{N}$, and $\bot$ for the smallest possible mask $\emptyset$. We will write $\Rrightarrow_{\mathcal{E}} P$ for ${}^{\mathcal{E}}\!\Rrightarrow^{\mathcal{E}} P$. Fancy updates satisfy the following basic proof rules:

FUP-MONO
$$\frac{P \vdash Q}{{}^{\mathcal{E}_1}\!\Rightarrow^{\mathcal{E}_2} P \vdash {}^{\mathcal{E}_1}\!\Rightarrow^{\mathcal{E}_2} Q}$$

FUP-INTRO-MASK
$$\frac{\mathcal{E}_2 \subseteq \mathcal{E}_1}{P \vdash {}^{\mathcal{E}_1}\!\Rightarrow^{\mathcal{E}_2} {}^{\mathcal{E}_2}\!\Rightarrow^{\mathcal{E}_1} P}$$

FUP-TRANS
$${}^{\mathcal{E}_1}\!\Rightarrow^{\mathcal{E}_2} {}^{\mathcal{E}_2}\!\Rightarrow^{\mathcal{E}_3} P \vdash {}^{\mathcal{E}_1}\!\Rightarrow^{\mathcal{E}_3} P$$

FUP-UPD
$$\dot{\Rightarrow} P \vdash \Rightarrow_{\mathcal{E}} P$$

FUP-FRAME
$$Q * {}^{\mathcal{E}_1}\!\Rightarrow^{\mathcal{E}_2} P \vdash {}^{\mathcal{E}_1 \uplus \mathcal{E}_\mathrm{f}}\!\Rightarrow^{\mathcal{E}_2 \uplus \mathcal{E}_\mathrm{f}} Q * P$$

FUP-UPDATE
$$\frac{a \rightsquigarrow \Phi}{\mathsf{Own}\,(a) \vdash \Rightarrow_{\mathcal{E}} \exists b.\, \Phi(b) \wedge \mathsf{Own}\,(b)}$$

FUP-TIMELESS
$$\frac{\mathsf{timeless}(P)}{\triangleright P \vdash \Rightarrow_{\mathcal{E}} P}$$

(There are no rules related to invariants here. Those rules will be discussed in the following subsection.)

We can further define the notions of *view shifts* and *linear view shifts*:

$$P \;{}^{\mathcal{E}_1}\!\Rightarrow\!\!\!\!\!\!*^{\mathcal{E}_2}\; Q \triangleq P \mathrel{-\!\!*} {}^{\mathcal{E}_1}\!\Rightarrow^{\mathcal{E}_2} Q$$
$$P \;{}^{\mathcal{E}_1}\!\Rightarrow^{\mathcal{E}_2}\; Q \triangleq \Box(P \mathrel{-\!\!*} {}^{\mathcal{E}_1}\!\Rightarrow^{\mathcal{E}_2} Q)$$
$$P \Rightarrow_{\mathcal{E}} Q \triangleq P \;{}^{\mathcal{E}}\!\Rightarrow^{\mathcal{E}}\; Q$$

These two are useful when writing down specifications and for comparing with previous versions of Iris, but for reasoning, it is typically easier to just work directly with fancy updates. Still, just to give an idea of what view shifts "are", here are some proof rules for them:

VS-UPDATE
$$\frac{a \rightsquigarrow \Phi}{\boxed{a}^{\gamma} \Rightarrow_{\emptyset} \exists b.\, \Phi(b) \wedge \boxed{b}^{\gamma}}$$

VS-TRANS
$$\frac{P \;{}^{\mathcal{E}_1}\!\Rightarrow^{\mathcal{E}_2}\; Q \qquad Q \;{}^{\mathcal{E}_2}\!\Rightarrow^{\mathcal{E}_3}\; R}{P \;{}^{\mathcal{E}_1}\!\Rightarrow^{\mathcal{E}_3}\; R}$$

VS-IMP
$$\frac{\Box\,(P \Rightarrow Q)}{P \Rightarrow_{\emptyset} Q}$$

VS-MASK-FRAME
$$\frac{P \;{}^{\mathcal{E}_1}\!\Rightarrow^{\mathcal{E}_2}\; Q}{P \;{}^{\mathcal{E}_1 \uplus \mathcal{E}'}\!\Rightarrow^{\mathcal{E}_2 \uplus \mathcal{E}'}\; Q}$$

VS-FRAME
$$\frac{P \;{}^{\mathcal{E}_1}\!\Rightarrow^{\mathcal{E}_2}\; Q}{P * R \;{}^{\mathcal{E}_1}\!\Rightarrow^{\mathcal{E}_2}\; Q * R}$$

VS-TIMELESS
$$\frac{\mathsf{timeless}(P)}{\triangleright P \Rightarrow_{\emptyset} P}$$

VS-DISJ
$$\frac{P \;{}^{\mathcal{E}_1}\!\Rightarrow^{\mathcal{E}_2}\; R \qquad Q \;{}^{\mathcal{E}_1}\!\Rightarrow^{\mathcal{E}_2}\; R}{P \vee Q \;{}^{\mathcal{E}_1}\!\Rightarrow^{\mathcal{E}_2}\; R}$$

VS-EXIST
$$\frac{\forall x.\, (P \;{}^{\mathcal{E}_1}\!\Rightarrow^{\mathcal{E}_2}\; Q)}{(\exists x.\, P) \;{}^{\mathcal{E}_1}\!\Rightarrow^{\mathcal{E}_2}\; Q}$$

VS-ALWAYS
$$\frac{\Box Q \vdash P \;{}^{\mathcal{E}_1}\!\Rightarrow^{\mathcal{E}_2}\; R}{P \wedge \Box Q \;{}^{\mathcal{E}_1}\!\Rightarrow^{\mathcal{E}_2}\; R}$$

VS-FALSE
$$\mathsf{False} \;{}^{\mathcal{E}_1}\!\Rightarrow^{\mathcal{E}_2}\; P$$

7.3 Invariant Namespaces

In §7.1, we defined a proposition $\boxed{P}^{\iota}$ expressing knowledge (*i.e.,* the proposition is persistent) that P is maintained as invariant with name ι. The concrete name ι is picked when the invariant is allocated, so it cannot possibly be statically known – it will always be a variable that's threaded through everything. However, we hardly care about the actual, concrete name. All we need to know is that this name is *different* from the names of other invariants that we want to open at the same time. Keeping track of the n^2 mutual inequalities that arise with n invariants quickly gets in the way of the actual proof.

To solve this issue, instead of remembering the exact name picked for an invariant, we will keep track of the *namespace* the invariant was allocated in. Namespaces are sets of invariants, following a tree-like structure: Think of the name of an invariant as a sequence of identifiers, much like a fully qualified Java class name. A *namespace* $\mathcal{N}$ then is like a Java package: it is a sequence of identifiers that we think of as *containing* all invariant names that begin with this sequence. For example, `conf.pldi.transfiris` is a namespace containing the invariant name `conf.pldi.transfiris.heap`.

The crux is that all namespaces contain infinitely many invariants, and hence we can *freely pick* the namespace an invariant is allocated in – no further, unpredictable choice has to be made. Furthermore, we will often know that namespaces are *disjoint* just by looking at them. The namespaces $\mathcal{N}.\texttt{iris}$ and $\mathcal{N}.\texttt{gps}$ are disjoint no matter the choice of $\mathcal{N}$. As a result, there is often no need to track disjointness of namespaces, we just have to pick the namespaces that we allocate our invariants in accordingly.

Formally speaking, let $\mathcal{N} \in \mathsf{InvNamesp} \triangleq \mathit{List}(\mathbb{N})$ be the type of *invariant namespaces*. We use the notation $\mathcal{N}.\iota$ for the namespace $[\iota] +\!\!+ \mathcal{N}$. (In other words, the list is "backwards". This is because cons-ing to the list, like the dot does above, is easier to deal with in Coq than appending at the end.)

The elements of a namespaces are *structured invariant names* (think: Java fully qualified class name). They, too, are lists of $\mathbb{N}$, the same type as namespaces. In order to connect this up to the definitions of §7.1, we need a way to map structured invariant names to *InvName*, the type of "plain" invariant names. Any injective mapping namesp_inj will do; and such a mapping has to exist because $\mathit{List}(\mathbb{N})$ is countable and *InvName* is infinite. Whenever needed, we (usually implicitly) coerce $\mathcal{N}$ to its encoded suffix-closure, *i.e.,* to the set of encoded structured invariant names contained in the namespace:

$$\mathcal{N}^{\uparrow} \triangleq \{\iota \mid \exists \mathcal{N}'.\, \iota = \mathsf{namesp\_inj}(\mathcal{N}' +\!\!+ \mathcal{N})\}$$

We will overload the notation for invariant propositions for using namespaces instead of names:

$$\boxed{P}^{\mathcal{N}} \triangleq \exists \iota \in \mathcal{N}^{\uparrow}.\, \boxed{P}^{\iota}$$

We can now derive the following rules (this involves unfolding the definition of fancy updates):

INV-PERSIST
$$\boxed{P}^{\mathcal{N}} \vdash \Box \boxed{P}^{\mathcal{N}}$$

INV-ALLOC
$$\triangleright P \vdash \Rrightarrow_{\emptyset} \boxed{P}^{\mathcal{N}}$$

INV-OPEN
$$\frac{\mathcal{N} \subseteq \mathcal{E}}{\boxed{P}^{\mathcal{N}} \;\; \mathcal{E} \Rrightarrow^{\mathcal{E}\setminus\mathcal{N}} \triangleright P * (\triangleright P \;{}^{\mathcal{E}\setminus\mathcal{N}}\!\!\Rrightarrow\!\!\!\!\ast^{\mathcal{E}}\; \mathsf{True})}$$

INV-OPEN-TIMELESS
$$\frac{\mathcal{N} \subseteq \mathcal{E} \qquad \mathsf{timeless}(P)}{\boxed{P}^{\mathcal{N}} \;\; \mathcal{E} \Rrightarrow^{\mathcal{E}\setminus\mathcal{N}} P * (P \;{}^{\mathcal{E}\setminus\mathcal{N}}\!\!\Rrightarrow\!\!\!\!\ast^{\mathcal{E}}\; \mathsf{True})}$$

7.4 Non-atomic ("Thread-Local") Invariants

Sometimes it is necessary to maintain invariants that we need to open non-atomically, for instance when we are working in a sequential setting and do not care about functions being thread-safe. Clearly, for this mechanism to be sound we need something that prevents us from opening the same invariant twice, something like the masks that avoid reentrancy on the "normal", atomic invariants. The idea is to use tokens[19] that guard access to non-atomic invariants. Having the token $[\mathrm{NaInv} : p.\mathcal{E}]$ indicates that we can open all invariants in $\mathcal{E}$ non-atomically. The p here is the name of the *invariant pool*. This mechanism allows us to have multiple, independent pools of invariants that all have their own namespaces.

One way to think about non-atomic invariants is as "thread-local invariants", where every pool is a thread. Every thread thus has its own, independent set of invariants. Every thread threads through all the tokens for its own pool, so that each invariant can only be opened in the thread it belongs to. As a consequence, they can be kept open around any sequence of expressions (*i.e.,* there is no restriction to atomic expressions) – after all, there cannot be any races with other threads.

[19] Very much like the tokens that are used to encode "normal", atomic invariants

Concretely, this is the monoid structure we need:

$$\mathit{PId} \triangleq \mathit{GName}$$
$$\textsc{NaTok} \triangleq \wp^{\text{fin}}(\mathit{InvName}) \times \wp(\mathit{InvName})$$

For every pool, there is a set of tokens designating which invariants are *enabled* (closed). This corresponds to the mask of "normal" invariants. We re-use the structure given by namespaces for non-atomic invariants. Furthermore, there is a *finite* set of invariants that is *disabled* (open).

Owning tokens is defined as follows:

$$[\text{NaInv} : p.\mathcal{E}] \triangleq \boxed{(\emptyset, \mathcal{E})}^{p}$$
$$[\text{NaInv} : p] \triangleq [\text{NaInv} : p.\top]$$

Next, we define non-atomic invariants. To simplify this construction,we piggy-back into "normal" invariants.

$$\text{NaInv}^{p.\mathcal{N}}(P) \triangleq \exists \iota \in \mathcal{N}.\ \boxed{P * \boxed{(\{\iota\}, \emptyset)}^{p} \lor [\text{NaInv} : p.\ \{\iota\}\,]}^{\mathcal{N}}$$

We easily obtain:

NAInv-new-pool
$\text{True} \Rrightarrow_{\perp} \exists p.\ [\text{NaInv} : p]$

NAInv-tok-split
$[\text{NaInv} : p.\mathcal{E}_1 \uplus \mathcal{E}_2] \Leftrightarrow [\text{NaInv} : p.\mathcal{E}_1] * [\text{NaInv} : p.\mathcal{E}_2]$

NAInv-new-inv
$\triangleright P \Rrightarrow_{\mathcal{N}} \square\, \text{NaInv}^{p.\mathcal{N}}(P)$

NAInv-acc-open-timeless
$$\frac{\text{timeless}(P) \qquad \mathcal{N}^{\uparrow} \subseteq \mathcal{E} \qquad \mathcal{N}^{\uparrow} \subseteq \mathcal{F}}{\text{NaInv}^{p.\mathcal{N}}(P) * [\text{NaInv} : p.\mathcal{F}] \vdash \Rrightarrow_{\mathcal{E}} \triangleright(P * [\text{NaInv} : p.\mathcal{F} \setminus \mathcal{N}^{\uparrow}] * (\triangleright P * [\text{NaInv} : p.\mathcal{F} \setminus \mathcal{N}^{\uparrow}] \Rrightarrow\!\!\!\ast_{\mathcal{E}} [\text{NaInv} : p.\mathcal{F}]))}$$

NAInv-acc-open
$$\frac{\mathcal{N}^{\uparrow} \subseteq \mathcal{E} \qquad \mathcal{N}^{\uparrow} \subseteq \mathcal{F} \qquad \mathcal{I} \text{ satisfies the finite bounded existential property}}{\text{NaInv}^{p.\mathcal{N}}(P) * [\text{NaInv} : p.\mathcal{F}] \vdash \Rrightarrow_{\mathcal{E}} \triangleright(P * [\text{NaInv} : p.\mathcal{F} \setminus \mathcal{N}^{\uparrow}] * (\triangleright P * [\text{NaInv} : p.\mathcal{F} \setminus \mathcal{N}^{\uparrow}] \Rrightarrow\!\!\!\ast_{\mathcal{E}} [\text{NaInv} : p.\mathcal{F}]))}$$

While the last two rules seem quite complicated, they essentially state the following: if we know that an invariant on P holds, then we open it to obtain P and the other invariants. Then, if we can return P and the assertion about the other invariants, we can restore the invariants we had intially. We note that the last two rules differ slightly (namely in the placement of the later modality) from the rules obtained in finite Iris.

7.5 Satisfiability

We define

$$\text{satisfiable\_at}\ \mathcal{E}\ P \triangleq \text{satisfiable}(W * \boxed{\mathcal{E}}^{\gamma_{\text{EN}}} * P)$$

for a notion of satisfiability which allows us to interact with fancy updates (and by extension invariants). The meta-level proposition satisfiable_at $\mathcal{E}$ P means that P is satisfiable and the current mask is $\mathcal{E}$. At mask $\mathcal{E}$, the invariants in $\mathcal{E}$ can be opened. Specifically, we have the update rule:

$$\frac{\text{satisfiable\_at}\ \mathcal{E}_1\ ({}^{\mathcal{E}_1}\!\Rrightarrow^{\mathcal{E}_2} P)}{\text{satisfiable\_at}\ \mathcal{E}_2\ P}$$

Apart from that, all the relevant rules of satisfiable lift to satisfiable_at $\mathcal{E}$:

$$\frac{\text{satisfiable\_at } \mathcal{E}\ P \qquad P \vdash Q}{\text{satisfiable\_at } \mathcal{E}\ Q} \qquad\qquad \frac{\text{satisfiable\_at } \mathcal{E}\ (\triangleright P)}{\text{satisfiable\_at } \mathcal{E}\ P}$$

$$\frac{\text{satisfiable\_at } \mathcal{E}\ (P \lor Q) \qquad \mathcal{I} \text{ enjoys the finite existential property}}{\text{satisfiable\_at } \mathcal{E}\ P \lor \text{satisfiable\_at } \mathcal{E}\ Q}$$

$$\frac{\text{satisfiable\_at } \mathcal{E}\ (\exists x : A.\ \phi\, x) \qquad \mathcal{I} \text{ enjoys the small existential property} \qquad A \text{ is small}}{\exists x : A.\ \text{satisfiable\_at } \mathcal{E}\ (\phi\, x)}$$

7.6 Logical Steps

It will be useful to have a modality which allows to perform updates, akin to a fancy update, as well as to take steps using a later. We will call this modality *logical steps*.

The idea for the definition is to essentially alternate an arbitrary number of fancy updates and laters in a careful way.

We first define the auxiliary *eventually* modality which stays at one mask $\mathcal{E}$ while interleaving laters and fancy updates.

$$\langle \mathcal{E} \rangle_0 P \triangleq \Rrightarrow_{\mathcal{E}} P$$
$$\langle \mathcal{E} \rangle_{(1+n)} P \triangleq \Rrightarrow_{\mathcal{E}} \triangleright \Rrightarrow_{\mathcal{E}} \langle \mathcal{E} \rangle_n P$$

$$\langle \mathcal{E} \rangle P \triangleq \Rrightarrow_{\mathcal{E}} \exists n.\ \langle \mathcal{E} \rangle_n P$$

As we will never use eventually on its own, but rather use at as a component for defining logical steps, we do not provide any rules.

For a logical step, we first open up all the invariants from a mask $\mathcal{E}_1$, use eventually, and finally close the invariants again, shifting to another mask $\mathcal{E}_2$.

$$\mathcal{E}_1 \models\!\!\Longrightarrow^{\mathcal{E}_2}_n P \triangleq \mathcal{E}_1 \Rrightarrow^{\emptyset} \langle \emptyset \rangle_n {}^{\emptyset}\!\Rrightarrow^{\mathcal{E}_2} P$$
$$\mathcal{E}_1 \models\!\!\Longrightarrow^{\mathcal{E}_2} P \triangleq \mathcal{E}_1 \Rrightarrow^{\emptyset} \langle \emptyset \rangle {}^{\emptyset}\!\Rrightarrow^{\mathcal{E}_2} P$$

We can obtain the following properties:

LSTEP-FUPD-LEFT
$$\mathcal{E}_1 \Rrightarrow^{\mathcal{E}_2} \mathcal{E}_2 \models\!\!\Longrightarrow^{\mathcal{E}_3} P \vdash \mathcal{E}_1 \models\!\!\Longrightarrow^{\mathcal{E}_3} P$$

LSTEP-FUPD-RIGHT
$$\mathcal{E}_1 \models\!\!\Longrightarrow^{\mathcal{E}_2} \mathcal{E}_2 \Rrightarrow^{\mathcal{E}_3} P \vdash \mathcal{E}_1 \models\!\!\Longrightarrow^{\mathcal{E}_3} P$$

LSTEP-SQUASH
$$\mathcal{E}_1 \models\!\!\Longrightarrow^{\mathcal{E}_2} \triangleright P \vdash \mathcal{E}_1 \models\!\!\Longrightarrow^{\mathcal{E}_2} P$$

LSTEPN-LSTEP
$$\mathcal{E}_1 \models\!\!\Longrightarrow^{\mathcal{E}_2}_n P \vdash \mathcal{E}_1 \models\!\!\Longrightarrow^{\mathcal{E}_2} P$$

LSTEPN-LATER
$$\triangleright\, \mathcal{E}_1 \models\!\!\Longrightarrow^{\mathcal{E}_2}_n P \vdash \mathcal{E}_1 \models\!\!\Longrightarrow^{\mathcal{E}_2}_{(1+n)} P$$

LSTEPN-INTRO
$$\mathcal{E}_1 \Rrightarrow^{\mathcal{E}_2} P \vdash \mathcal{E}_1 \models\!\!\Longrightarrow^{\mathcal{E}_2}_0 P$$

LSTEPN-STEP-MONO
$$\frac{k_1 \le k_2}{\mathcal{E}_1 \models\!\!\Longrightarrow^{\mathcal{E}_2}_{k_1} P \vdash \mathcal{E}_1 \models\!\!\Longrightarrow^{\mathcal{E}_2}_{k_2} P}$$

LSTEP-MONO
$$P \mathrel{-\!\!*} Q \vdash \mathcal{E}_1 \models\!\!\Longrightarrow^{\mathcal{E}_2} P \mathrel{-\!\!*} \mathcal{E}_1 \models\!\!\Longrightarrow^{\mathcal{E}_2} Q$$

LSTEPN-MONO
$$P \mathrel{-\!\!*} Q \vdash \mathcal{E}_1 \models\!\!\Longrightarrow^{\mathcal{E}_2}_n P \mathrel{-\!\!*} \mathcal{E}_1 \models\!\!\Longrightarrow^{\mathcal{E}_2}_n Q$$

8 Languages

A *language* Λ consists of a set *Expr* of *expressions* (metavariable e), a set *Val* of *values* (metavariable v), a set *Obs* of *observations*[20] (or "observable events") and a set *State* of *states* (metavariable σ) such that

- There exist functions $\mathrm{val\_to\_expr} : \mathit{Val} \to \mathit{Expr}$ and $\mathrm{expr\_to\_val} : \mathit{Expr} \rightharpoonup \mathit{Val}$ (notice the latter is partial), such that

$$\forall e, v.\, \mathrm{expr\_to\_val}(e) = v \Rightarrow \mathrm{val\_to\_expr}(v) = e \qquad \forall v.\, \mathrm{expr\_to\_val}(\mathrm{val\_to\_expr}(v)) = v$$

- There exists a *primitive reduction relation*

$$(-, - \xrightarrow{-}_{\mathtt{t}} -, -, -) \subseteq (\mathit{Expr} \times \mathit{State}) \times \mathit{List}(\mathit{Obs}) \times (\mathit{Expr} \times \mathit{State} \times \mathit{List}(\mathit{Expr}))$$

  A reduction $e_1, \sigma_1 \xrightarrow{\vec{\kappa}}_{\mathtt{t}} e_2, \sigma_2, \vec{e}$ indicates that, when e_1 in state σ_1 reduces to e_2 with new state σ_2, the new threads in the list $\vec{e}$ is forked off and the observations $\vec{\kappa}$ are made. We will write $e_1, \sigma_1 \to_{\mathtt{t}} e_2, \sigma_2$ for $e_1, \sigma_1 \xrightarrow{()}_{\mathtt{t}} e_2, \sigma_2, ()$, *i.e.,* when no threads are forked off and no observations are made.

- All values are stuck:

$$e, \_ \to_{\mathtt{t}} \_, \_, \_ \Rightarrow \mathrm{expr\_to\_val}(e) = \bot$$

Definition 24. *An expression e and state σ are* reducible *(written $\mathrm{red}(e, \sigma)$) if*

$$\exists \vec{\kappa}, e_2, \sigma_2, \vec{e}.\, e, \sigma \xrightarrow{\vec{\kappa}}_{\mathtt{t}} e_2, \sigma_2, \vec{e}$$

Definition 25. *An expression e is* strongly atomic *if it reduces in one step to a value:*

$$\mathrm{strongly\_atomic}(e) \triangleq \forall \sigma_1, \vec{\kappa}, e_2, \sigma_2, \vec{e}.\, e, \sigma_1 \xrightarrow{\vec{\kappa}}_{\mathtt{t}} e_2, \sigma_2, \vec{e} \Rightarrow \mathrm{expr\_to\_val}(e_2) \neq \bot$$

Definition 26 (Context)**.** *A function $K : \mathit{Expr} \to \mathit{Expr}$ is a* context *if the following conditions are satisfied:*

1. *K does not turn non-values into values:*

$$\forall e.\, \mathrm{expr\_to\_val}(e) = \bot \Rightarrow \mathrm{expr\_to\_val}(K(e)) = \bot$$

2. *One can perform reductions below K:*

$$\forall e_1, \sigma_1, \vec{\kappa}, e_2, \sigma_2, \vec{e}.\, e_1, \sigma_1 \xrightarrow{\vec{\kappa}}_{\mathtt{t}} e_2, \sigma_2, \vec{e} \Rightarrow K(e_1), \sigma_1 \xrightarrow{\vec{\kappa}}_{\mathtt{t}} K(e_2), \sigma_2, \vec{e}$$

3. *Reductions stay below K until there is a value in the hole:*

$$\forall e_1', \sigma_1, \vec{\kappa}, e_2, \sigma_2, \vec{e}.\, \mathrm{expr\_to\_val}(e_1') = \bot \wedge K(e_1'), \sigma_1 \xrightarrow{\vec{\kappa}}_{\mathtt{t}} e_2, \sigma_2, \vec{e} \Rightarrow$$
$$\exists e_2'.\, e_2 = K(e_2') \wedge e_1', \sigma_1 \xrightarrow{\vec{\kappa}}_{\mathtt{t}} e_2', \sigma_2, \vec{e}$$

[20] See https://gitlab.mpi-sws.org/iris/iris/merge_requests/173 for how observations are useful to encode prophecy variables.

8.1 Concurrent Language

For any language Λ, we define the corresponding thread-pool semantics.

Machine syntax

$$T \in \mathit{ThreadPool} \triangleq \mathit{List}(\mathit{Expr})$$

Machine reduction $\boxed{T; \sigma \xrightarrow{\vec{\kappa}}_{\mathsf{tp}} T'; \sigma'}$

$$\frac{e_1, \sigma_1 \xrightarrow{\vec{\kappa}}_{\mathsf{t}} e_2, \sigma_2, \vec{e}}{T +\!\!+ [e_1] +\!\!+ T'; \sigma_1 \xrightarrow{\vec{\kappa}}_{\mathsf{tp}} T +\!\!+ [e_2] +\!\!+ T' +\!\!+ \vec{e}; \sigma_2}$$

We use $\xrightarrow{-}{}^{*}_{\mathsf{tp}}$ for the reflexive transitive closure of $\xrightarrow{-}_{\mathsf{tp}}$, as usual concatenating the lists of observations of the individual steps. For the case that we do not care about the observations, we define

$$T; \sigma \to_{\mathsf{tp}} T'; \sigma' \triangleq \exists \vec{\kappa}. T; \sigma \xrightarrow{\vec{\kappa}}_{\mathsf{tp}} T'; \sigma'.$$

9 Program Logic

We now describe how we can build a program logic for an arbitrary language as defined in Section 8 on top of the Transfinite Iris base logic with invariants and higher-order ghost state.

Fix some language Λ for the rest of this section. We assume that everything making up the definition of the language, *i.e.,* values, expressions, states, the conversion functions, reduction relation and all their properties, are suitably reflected into the logic.

Weakest preconditions are the core piece of the program logic, allowing us to reason about program behaviour. We define several variations of them: first an adaption of the standard finite Iris weakest precondition allowing to reason about *safety*; secondly a weakest precondition for termination-preserving refinement; and finally, we can derive a weakest precondition ensuring termination as a special case of the refinement weakest preconditon.

9.1 Weakest Precondition for Safety

In principle, we could leave the definition of weakest preconditions unchanged from finite Iris for reasoning about safety: the minor changes to the base logic do not invalidate the definition. However, its usefulness would be severely restricted when working with invariants. Due to our definition of invariants involving higher-order ghost state, we can only get the contents of an invariant under a later when opening it. In finite Iris, we could then use the commuting properties for later with separating conjunctions and existentials to still directly access some timeless parts of an invariant. Many existing Iris developments rely on this pattern for their choice of invariants.

As we have seen, we cannot get these commuting properties in Transfinite Iris. In order to mitigate this loss, we change the definition of weakest preconditions: we move the later modality we obtain when proving a weakest precondition, so that, when proving a weakest precondition, we get the later *before* we have to prove a step, instead of *after*. To make porting proofs easier, we do however also keep the later at the existing position.

Additionally, as a benefit of our transfinite model, we can actually generalize the single later modality of Iris to obtain an *arbitrary* number of laters (existentially quantified) in each step. Therefore, our definition will add a logical step as defined in Section 7.6 to the definition[21].

Defining the weakest precondition The weakest precondition is parameterized over a *state interpretation* $S : \mathit{State} \times \mathit{List}(\mathit{Obs}) \times \mathbb{N} \to \mathit{iProp}$ that interprets the execution state as an Iris proposition, and a predicate $\Phi_F : \mathit{Val} \to \mathit{iProp}$ serving as a postcondition for forked-off threads. The state interpretation can depend on the current physical state, the list of *future* observations as well as the total number of *forked* threads (that is one less that the total number of threads). This can be instantiated, for example, with ownership of an authoritative RA to tie the physical state to fragments that are used for user-level proofs. Finally, the weakest precondition takes a parameter $s \in \{\mathsf{NotStuck}, \mathsf{Stuck}\}$ indicating whether program execution is allowed to get stuck.

We now provide the definition of the weakest precondition, with the only change being that we change the fancy update for opening the invariants to a logical step (Section 7.6), which allows us to obtain an

[21] Currently, this generalization is not needed for any of the safety reasoning proofs we ported to Transfinite Iris.

arbitrary number of laters, highlighted in blue:

$$\begin{aligned}
\mathit{wp}(S, \Phi_F, s) \triangleq \mu\ \mathit{wp_rec}.\ \lambda\, \mathcal{E}, e, \Phi.\ & \\
& (\exists v.\, \mathrm{expr_to_val}(e) = v \land \Rightarrow\!\!\!\!\Rightarrow_{\mathcal{E}} \Phi(v)) \lor \\
& \Big(\mathrm{expr_to_val}(e) = \bot \land \forall \sigma, \vec{\kappa}, \vec{\kappa}', n.\, S(\sigma, \vec{\kappa} +\!\!+ \vec{\kappa}', n) \twoheadrightarrow {}^{\mathcal{E}}\!\!\Longrightarrow^{\emptyset} \\
& \qquad (s = \mathsf{NotStuck} \Rightarrow \mathrm{red}(e, \sigma)) * \forall e', \sigma', \vec{e}.\, (e, \sigma \xrightarrow{\vec{\kappa}}_{\mathsf{t}} e', \sigma', \vec{e})\ {}^{\emptyset}\!\!\Rrightarrow^{\emptyset} \triangleright\, {}^{\emptyset}\!\!\Rightarrow\!\!\!\!\Rightarrow^{\mathcal{E}} \\
& \qquad\quad S(\sigma', \vec{\kappa}', n + |\vec{e}|) * \mathit{wp_rec}(\mathcal{E}, e', \Phi) * \textstyle\bigstar_{e'' \in \vec{e}}\ \mathit{wp_rec}(\top, e'', \Phi_F)\Big)
\end{aligned}$$

$$\mathsf{wp}^{S;\Phi_F}_{s;\mathcal{E}}\ e\ \{v.\ P\} \triangleq \mathit{wp}(S, \Phi_F, s)(\mathcal{E}, e, \lambda\, v.\ P)$$

We use Banach's fixpoint theorem (Theorem 1) in this definition. The S and Φ_F will always be set by the context; typically, when instantiating Iris with a language, we also pick the corresponding state interpretation S and fork-postcondition Φ_F. All proof rules leave S and Φ_F unchanged. If we leave away the mask $\mathcal{E}$, we assume it to default to $\top$. If we leave away the stuckness s, it defaults to NotStuck.

Note that we retain the later in the definition after proving a step for compatibility reasons: we could in principle remove it (as the logical steps already make the definition contractive), but that would require larger changes to existing proofs in finite Iris. The logical step will essentially allow us to pull out an arbitrary number of laters (that we however have to fix before starting to pull out laters) before proving that we can take a step, giving us a way to deal with the loss of commuting rules. Using logical steps instead of a single later is additionally not only useful for working around the commuting rules, but also for uses where multiple laters need to be eliminated, *e.g.,* for opening nested invariants in a single step of the program (which is currently not possible in finite Iris).

In order to actually facilitate this, we additionally define a *stronger weakest precondition* swp parameterized over a natural number k. Instead of allowing us to take an arbitrary number of laters (via the logical step), we can only take a previously chosen number of steps k. Additionally, the swp requires us to prove that we can actually take a step, omitting the case that we have already reached a value.

$$\begin{aligned}
\mathsf{swp}^{k;S;\Phi_F}_{s;\mathcal{E}}\ e\ \{v.\ P\} \triangleq\ & \forall \sigma, \vec{\kappa}, \vec{\kappa}', n.\, S(\sigma, \vec{\kappa} +\!\!+ \vec{\kappa}', n) \twoheadrightarrow {}^{\mathcal{E}}\!\!\Longrightarrow^{\emptyset}_{k} \\
& (s = \mathsf{NotStuck} \Rightarrow \mathrm{red}(e, \sigma)) * \forall e', \sigma', \vec{e}.\, (e, \sigma \xrightarrow{\vec{\kappa}}_{\mathsf{t}} e', \sigma', \vec{e})\ {}^{\emptyset}\!\!\Rrightarrow^{\emptyset} \triangleright\, {}^{\emptyset}\!\!\Rightarrow\!\!\!\!\Rightarrow^{\mathcal{E}} \\
& S(\sigma', \vec{\kappa}', n + |\vec{e}|) * \mathsf{wp}^{S;\Phi_F}_{s;\mathcal{E}}\ e'\ \{v.\ P\} * \textstyle\bigstar_{e'' \in \vec{e}}\ \mathsf{wp}^{S;\Phi_F}_{s;\top}\ e''\ \{\Phi_F\}
\end{aligned}$$

Laws of the weakest precondition The crucial new rule of this transfinite weakest precondition is that we can transition from a weakest precondition to the strong weakest precondition:

$$\textsc{swp-wp}\ \frac{\mathrm{expr\_to\_val}(e) = \bot}{\mathsf{swp}^{k}_{s;\mathcal{E}}\ e\ \{v.\ P\} \vdash \mathsf{wp}_{s;\mathcal{E}}\ e\ \{v.\ P\}}$$

As one would expect, the minor change of the definition does not impact the rules known from Iris much, with the only exception being WP-ATOMIC where we currently require strong atomicity independently

of the stuckness:

WP-VALUE
$$P[v/x] \vdash \mathsf{wp}_{s;\mathcal{E}}\, v\, \{x.\, P\}$$

WP-MONO
$$\frac{\mathcal{E}_1 \subseteq \mathcal{E}_2 \qquad \Gamma, x : \mathsf{val} \mid P \vdash \Rrightarrow_{\mathcal{E}_2} Q \qquad (s_2 = \mathsf{Stuck} \vee s_1 = s_2)}{\Gamma \mid \mathsf{wp}_{s_1;\mathcal{E}_1}\, e\, \{x.\, P\} \vdash \mathsf{wp}_{s_2;\mathcal{E}_2}\, e\, \{x.\, Q\}}$$

FUP-WP
$$\Rrightarrow_{\mathcal{E}} \mathsf{wp}_{s;\mathcal{E}}\, e\, \{x.\, P\} \vdash \mathsf{wp}_{s;\mathcal{E}}\, e\, \{x.\, P\}$$

WP-FUP
$$\mathsf{wp}_{s;\mathcal{E}}\, e\, \{x.\, \Rrightarrow_{\mathcal{E}} P\} \vdash \mathsf{wp}_{s;\mathcal{E}}\, e\, \{x.\, P\}$$

WP-ATOMIC
$$\frac{\mathrm{strongly\_atomic}(e)}{{}^{\mathcal{E}_1}\Rrightarrow^{\mathcal{E}_2} \mathsf{wp}_{s;\mathcal{E}_2}\, e \left\{x.\, {}^{\mathcal{E}_2}\Rrightarrow^{\mathcal{E}_1} P\right\} \vdash \mathsf{wp}_{s;\mathcal{E}_1}\, e\, \{x.\, P\}}$$

WP-FRAME
$$Q * \mathsf{wp}_{s;\mathcal{E}}\, e\, \{x.\, P\} \vdash \mathsf{wp}_{s;\mathcal{E}}\, e\, \{x.\, Q * P\}$$

WP-FRAME-STEP
$$\frac{\mathrm{expr\_to\_val}(e) = \bot \qquad \mathcal{E}_2 \subseteq \mathcal{E}_1}{\mathsf{wp}_{s;\mathcal{E}_2}\, e\, \{x.\, P\} * {}^{\mathcal{E}_1}\Rrightarrow^{\mathcal{E}_2} \triangleright\, {}^{\mathcal{E}_2}\Rrightarrow^{\mathcal{E}_1} Q \vdash \mathsf{wp}_{s;\mathcal{E}_1}\, e\, \{x.\, Q * P\}}$$

WP-BIND
$$\frac{K \text{ is a context}}{\mathsf{wp}_{s;\mathcal{E}}\, e \left\{x.\, \mathsf{wp}_{s;\mathcal{E}}\, K(\mathrm{val\_to\_expr}(x))\, \{y.\, P\}\right\} \vdash \mathsf{wp}_{s;\mathcal{E}}\, K(e)\, \{y.\, P\}}$$

We will also want a rule that connect weakest preconditions to the operational semantics of the language. This basically just copies the second branch (the non-value case) of the definition of weakest preconditions, but without offering the logical step.

WP-LIFT-STEP
$$\frac{\mathrm{expr\_to\_val}(e_1) = \bot}{\begin{array}{l}\forall \sigma_1, \vec{\kappa}, \vec{\kappa}', n.\, S(\sigma_1, \vec{\kappa} +\!\!+ \vec{\kappa}', n) \;{}^{\mathcal{E}}\!\Rrightarrow\!\!\!\!\ast^{\emptyset}\; (s = \mathsf{NotStuck} \Rightarrow \mathrm{red}(e_1, \sigma_1)) * \\ \quad \forall e_2, \sigma_2, \vec{e}.\, (e_1, \sigma_1 \xrightarrow{\vec{\kappa}}_{\mathsf{t}} e_2, \sigma_2, \vec{e}) \;{}^{\emptyset}\!\Rrightarrow\!\!\!\!\ast^{\emptyset}\; \triangleright\, {}^{\emptyset}\!\Rrightarrow^{\mathcal{E}} \\ \qquad \left(S(\sigma_2, \vec{\kappa}', n + |\vec{e}|) * \mathsf{wp}^{S;\Phi_F}_{s;\mathcal{E}}\, e_2\, \{x.\, P\} * \ast_{e_{\mathrm{f}} \in \vec{e}}\, \mathsf{wp}^{S\Phi_F}_{s;\top}\, e_{\mathrm{f}}\, \{\Phi_F\}\right) \\ \vdash \mathsf{wp}^{S;\Phi_F}_{s;\mathcal{E}}\, e_1\, \{x.\, P\}\end{array}}$$

Laws of the strong weakest precondition Very similar laws hold for the strong weakest precondition, with the outstanding rule being the following one, allowing us to pull out a later while decreasing the index:

SWP-DO-STEP
$$\triangleright\, \mathsf{swp}^{k}_{s;\mathcal{E}}\, e\, \{x.\, P\} \vdash \mathsf{swp}^{(1+k)}_{s;\mathcal{E}}\, e\, \{x.\, P\}$$

SWP-MONO
$$\frac{\mathcal{E}_1 \subseteq \mathcal{E}_2 \qquad \Gamma, x : \mathsf{val} \mid P \vdash \Rrightarrow_{\mathcal{E}_2} Q \qquad (s_2 = \mathsf{Stuck} \vee s_1 = s_2)}{\Gamma \mid \mathsf{swp}^k_{s_1;\mathcal{E}_1}\, e\, \{x.\, P\} \vdash \mathsf{swp}^k_{s_2;\mathcal{E}_2}\, e\, \{x.\, Q\}}$$

FUP-SWP
$$\Rrightarrow_{\mathcal{E}} \mathsf{swp}^k_{s;\mathcal{E}}\, e\, \{x.\, P\} \vdash \mathsf{swp}^k_{s;\mathcal{E}}\, e\, \{x.\, P\}$$

SWP-FUP
$$\mathsf{swp}^k_{s;\mathcal{E}}\, e\, \{x.\, \Rrightarrow_{\mathcal{E}} P\} \vdash \mathsf{swp}^k_{s;\mathcal{E}}\, e\, \{x.\, P\}$$

SWP-ATOMIC
$$\frac{\mathrm{strongly\_atomic}(e)}{{}^{\mathcal{E}_1}\Rrightarrow^{\mathcal{E}_2} \mathsf{swp}^k_{s;\mathcal{E}_2}\, e \left\{x.\, {}^{\mathcal{E}_2}\Rrightarrow^{\mathcal{E}_1} P\right\} \vdash \mathsf{swp}^k_{s;\mathcal{E}_1}\, e\, \{x.\, P\}}$$

SWP-FRAME
$$Q * \mathsf{swp}^k_{s;\mathcal{E}}\, e\, \{x.\, P\} \vdash \mathsf{swp}^k_{s;\mathcal{E}}\, e\, \{x.\, Q * P\}$$

SWP-BIND
$$\frac{K \text{ is a context}}{\mathsf{swp}^k_{s;\mathcal{E}}\, e \left\{x.\, \mathsf{wp}_{s;\mathcal{E}}\, K(\mathrm{val\_to\_expr}(x))\, \{y.\, P\}\right\} \vdash \mathsf{swp}^k_{s;\mathcal{E}}\, K(e)\, \{y.\, P\}}$$

SWP-LIFT-STEP
$$\begin{aligned}
&\forall \sigma_1, \vec{\kappa}, \vec{\kappa}', n.\, S(\sigma_1, \vec{\kappa} +\!\!+ \vec{\kappa}', n) \;{}^{\mathcal{E}}\!\Rrightarrow\!\!\!\!\ast^{\emptyset}\; (s = \mathsf{NotStuck} \Rightarrow \mathrm{red}(e_1, \sigma_1)) * \\
&\quad \forall e_2, \sigma_2, \vec{e}.\, (e_1, \sigma_1 \xrightarrow{\vec{\kappa}}_{\mathsf{t}} e_2, \sigma_2, \vec{e})\; {}^{\emptyset}\!\Rrightarrow\!\!\!\!\ast^{\emptyset} \triangleright\, {}^{\emptyset}\!\Rrightarrow^{\mathcal{E}} \\
&\qquad \left(S(\sigma_2, \vec{\kappa}', n + |\vec{e}|) * \mathsf{wp}^{S;\Phi_F}_{s;\mathcal{E}}\, e_2\, \{x.\, P\} * \ast_{e_{\mathrm{f}} \in \vec{e}}\, \mathsf{wp}^{S\Phi_F}_{s;\top}\, e_{\mathrm{f}}\, \{\Phi_F\}\right) \\
&\vdash \mathsf{swp}^{k;S;\Phi_F}_{s;\mathcal{E}}\, e_1\, \{x.\, P\}
\end{aligned}$$

Adequacy of the weakest precondition The purpose of the adequacy statement is to show that our notion of weakest preconditions is *realistic* in the sense that it actually has anything to do with the actual behavior of the program. Since we have only changed the definition of the weakest precondition for safety in minor ways, we can keep the adequacy theorem and change the proof to account for the added logical step[22]. This will however only enable the adequacy result for transfinite step-index types as we cannot obtain a soundness result for logical steps with finite index types.

Theorem 3 (Adequacy)**.** *Assume that the underlying step-index type is transfinite.*

Assume we are given some e_1, σ_1, $\vec{\kappa}$, T_2, σ_2 *such that* $([e_1], \sigma_1) \xrightarrow{\vec{\kappa}}{}^*_{\mathsf{tp}} (T_2, \sigma_2)$*, and we are also given a* meta-level *property* p *that we want to show. We assume that* p *has been suitably reflected into our logic so that we can talk about it inside Transfinite Iris.*

To verify that p *holds, it is sufficient to show the following Iris entailment:*

$$\mathit{True} \vdash \Rrightarrow_{\top} \exists s, S, \Phi, \Phi_F.\, S(\sigma_1, \vec{\kappa}, 0) * \mathit{wp}^{S;\Phi_F}_{s;\top}\, e_1\, \{x.\, \Phi(x)\} * \left(C^{S;\Phi;\Phi_F}_s(T_2, \sigma_2) \;\; {}^{\top}\!\Rrightarrow^{\emptyset}\; p\right)$$

[22] We do not currently use our new notion of satisfiability (see Section 4.4.1) to enable a modular proof. However, our adequacy proof for the refinement weakest precondition introduced later makes use of the new technique.

where C describes states that are consistent with the state interpretation and postconditions:

$$\begin{aligned} C_s^{S;\Phi;\Phi_F}(T_2, \sigma_2) \triangleq \exists e_2, T_2'.\, T_2 = [e_2] +\!\!+ T_2' * \\ (s = \mathit{NotStuck} \Rightarrow \forall e \in T_2.\, \mathrm{expr\_to\_val}(e) \neq \bot \vee \mathrm{red}(e, \sigma_2)) * \\ S(\sigma_2, (), |T_2'|) * \\ (\mathrm{expr\_to\_val}(e_2) \neq \bot \mathrel{-\!\!*} \Phi(\mathrm{expr\_to\_val}(e_2))) * \\ \left(\ast_{e \in T_2'} \mathrm{expr\_to\_val}(e) \neq \bot \mathrel{-\!\!*} \Phi_F(\mathrm{expr\_to\_val}(e))\right) \end{aligned}$$

In other words, to show that p holds, we have to prove an entailment in Iris that, starting from the empty context, chooses some state interpretation, postcondition, forked-thread postcondition and stuckness and then proves:

- the initial state interpretation,
- a weakest-precondition,
- and a view shift showing the desired p under the extra assumption $C(T_2, \sigma_2)$.

Notice that the state interpretation and the postconditions are chosen *after* doing a fancy update, which allows them to depend on the names of ghost variables that are picked in that initial fancy update. This gives us a chance to allocate some "global" ghost state that state interpretation and postcondition can refer to.

$C_s^{S;\Phi;\Phi_F}(T_2, \sigma_2)$ says that:

- The final thread-pool T_2 contains the final state of the main thread e_2, and any number of additional forked threads in T_2'.
- If this is a stuck-free weakest precondition, then all threads in the final thread-pool are either values or are reducible in the final state σ_2.
- The state interpretation S holds for the final state.
- If the main thread reduced to a value, the post-condition Φ of the weakest precondition holds for that value.
- If any other thread reduced to a value, the forked-thread post-condition Φ_F holds for that value.

As the crucial difference to the finite Iris adequacy proof, we use the following soundness result for logical steps:

Lemma 11 (Soundness of Logical Steps)**.** *Assume we are working with a transfinite step-index type. Let p be a meta-level property reflected into our logic.*

If $\mathit{True} \vdash (\,^{\top}\!\!\models\!\!\Rightarrow^{\top})^n\, p$*, then p holds.*

It is clear that this lemma cannot hold for finite index types as we know that $\vdash \exists n.\, \triangleright^n \bot$ is provable for natural number indices (as seen in Section 2.3).

9.2 Generalized Simulations

While Iris allows us to perform safety reasoning, the novelty of Transfinite Iris lies in providing liveness reasoning, justified by the existential property. An interesting class of liveness properties is provided by termination-preserving refinements and termination. Abstractly, a termination-preserving refinement between two terms s in a source language S and t in a target language T says that (1) if t evaluates to v, then s evaluates to a related value v', and (2) if s terminates, then t terminates.

Before getting to the actual definition of the refinement weakest precondition which is able to handle all features of the Iris program logic, we show at a high level how to generalize the simulation relation from the key ideas section of this work to handle stuttering steps of the source (explained below). This will then motivate some of the structure of our definition of the weakest precondition.

In the key ideas section, we have only given an internalized simulation $(\preceq_*)$ which establishes a "lock-step" simulation (*i.e.,* one step of the source for every step of the target) between the target and the source. Such a lock-step simulation is too strict for verifying examples such as memo_rec. Realistic examples require the ability to take steps in the target program without corresponding steps in the source program, also known as *stuttering*.

A traditional technique to enable stuttering is explicitly counting the *stutters* (*i.e.,* steps of the target where the source does not change as well) in the definition of the simulation relation (*e.g.,* as a subscript of the relation). Unfortunately, counting stutters easily becomes tedious in proofs because (1) the number of stutters has to be maintained explicitly, and (2) counting stutters is not very compositional (*e.g.,* for function calls in the target, the number of stutters can depend on the argument). Thus, we will use a form of stuttering which does not require explicit stutter counting for the simulation $(\preceq_*)$. To the best of our knowledge, this is the first simulation which combines guarded recursion (used for cyclic features of the language) and stuttering without explicit counting.

Now, in the absence of a step count, we have to resort to alternative measures to ensure that the stutters terminate; otherwise the target could diverge. We incorporate termination (without counting steps) in the definition using a *least fixpoint*. That is, we define:

$$t \preceq_* s \triangleq (\exists b.\, s = t = b) \vee \left(\begin{array}{l} (\exists t'.\, t \rightsquigarrow_{\text{tgt}} t') \wedge \forall t'.\, t \rightsquigarrow_{\text{tgt}} t' \Rightarrow \\ \quad t' \preceq_* s \vee \exists s'.\, s \rightsquigarrow_{\text{src}} s' \wedge \triangleright\, t' \preceq_* s' \end{array} \right)$$

as a least fixpoint. In this version of $(\preceq_*)$, the user is offered a choice after each target step $t \rightsquigarrow_{\text{tgt}} t'$: either she proves the simulation for the new target t', or she simulates the target step by picking one in the source $s \rightsquigarrow_{\text{src}} s'$ and then proving the simulation for $t' \preceq_* s'$.

Now, to understand this definition in more detail, we consider once more its unfolding in the step-indexed model[23]. In the model, the simulation relation is defined as:

$$t \preceq_0 s \triangleq \mathsf{True}$$

$$t \preceq_{i+1} s \triangleq (\exists b.\, t = s = b) \vee \left(\begin{array}{l} (\exists t'.\, t \rightsquigarrow_{\text{tgt}} t') \wedge \forall t'.\, t \rightsquigarrow_{\text{tgt}} t' \Rightarrow \\ \quad t' \preceq_{i+1} s \vee \exists s'.\, s \rightsquigarrow_{\text{src}} s' \wedge t' \preceq_i s' \end{array} \right)$$

[23] For simplicity, instead of showing the unfolding with ordinals as step-indices, we show the unfolding with natural numbers as step-indices (in the style of the key ideas section).

where for each i, the definition of $(\preceq_{i+1})$ is a *least fixpoint*. The use of the least fixpoint ensures that for each $(\preceq_{i+1})$, the left side of the disjunction can only be taken finitely often.

In proofs of $(\preceq_*)$, the choice between the left and right side of the disjunction can be made as follows. The right side should be chosen, if the next target step corresponds to a source step. The left side should be chosen, if the next target step or the next several target steps do not have a direct correspondence to source steps but have an intrinsic notion of termination. By intrinsic notion of termination, we mean that an inductive argument can be made why the execution terminates (*e.g.*, a function structurally recurses on a list, or a for-loop iterates for a fixed number of iterations).

To justify that the simulation relation defined above is actually useful, we show its *adequacy*, meaning we show it entails a termination preserving refinement.

Lemma 7. *If* $\vdash t \preceq_* s$*, then:*

- *for all* b*, if* t *evaluates to* b*, then* s *evaluates to* b*,*
- *if* t *diverges, then* s *diverges.*

Proof.

- First we prove the result refinement: First, we observe that for a single step $t \rightsquigarrow_{\text{tgt}} t'$, we obtain from $\vdash t \preceq_* s$:

  $$\vdash t' \preceq_* s \text{ or } (s \rightsquigarrow_{\text{src}} s' \text{ and } \vdash t' \preceq_* s' \text{ for some } s')$$

  by using the existential property and other soundness rules of the logic. In particular, this means:

  $$s \rightsquigarrow^*_{\text{src}} s' \text{ and } \vdash t' \preceq_* s' \text{ for some } s'$$

  where $(\rightsquigarrow^*_{\text{src}})$ is the reflexive, transitive closure of $(\rightsquigarrow_{\text{src}})$.

  By repeatedly applying this single step rule to the simulation $\vdash t \preceq_* s$ for every step in the execution of t to b, we obtain:

  $$s \rightsquigarrow^*_{\text{src}} s' \text{ and } \vdash b \preceq_* s' \text{ for some } s'$$

  By unfolding the definition of the simulation relation, we obtain $\vdash b = s'$ (since b cannot step), and hence $b = s'$.

- Now we show termination refinement: Let t by diverging. Then we can prove inside of the logic:

  $$\vdash \exists t', s'.\ t' \text{ diverges} \land s \rightsquigarrow_{\text{src}} s' \land \triangleright\, t' \preceq_* s'$$

  by induction on the least fixpoint $(\preceq_*)$, incrementally unrolling the infinite execution of t. Using the existential property and other standard soundness properties, we obtain:

  $$s \rightsquigarrow_{\text{src}} s' \text{ and } t' \text{ diverges and } \vdash t' \preceq_* s' \text{ for some } t', s'$$

  By repeating this step, starting at t', we can extract an infinite execution of s.

□

9.3 Refinement Weakest Precondition

With this motivation for the core style of reasoning we employ, we introduce a new type of weakest precondition. We follow the Iris style and setup the refinement weakest precondition in a very general way. Instead of specializing to one particular language, we parameterize over a suitable notion of *source language*. This will pay off later as it allows us to obtain a weakest precondition for termination as a special case. While we are reasoning with the weakest precondition in the *target language*, we have certain requirements on making steps in the source language.

9.3.1 Source languages

Definition 5 (Source language)**.** A source language consists of a type A, a relation $\hookrightarrow$ on A, and a source interpretation $I : A \to \mathit{iProp}$.

This definition of source languages is very general, with the source interpretation providing a great deal of flexibility.

Definition 6 (Source update)**.** It is possible to do a source update to $P : \mathit{iProp}$, written $\Rightarrow\!\!\!\!\!\!\Rightarrow^{s}_{\mathcal{E}} P$, if

$$\forall a : A. I(a) \Rrightarrow\!\!\!\!\!\!\Rightarrow_{\mathcal{E}} \exists b : A.\, a \hookrightarrow^{+} b * I(b) * P.$$

Note that P may very well make assertions about the state of the source language if we tie it to the source interpretation with suitable ghost state.

We have the following notable rules for source updates:

SRC-UPDATE-BIND
$\Rightarrow\!\!\!\!\!\!\Rightarrow^{s}_{\mathcal{E}} P * (P \mathrel{-\!\!*} \Rightarrow\!\!\!\!\!\!\Rightarrow^{s}_{\mathcal{E}} Q) \vdash \Rightarrow\!\!\!\!\!\!\Rightarrow^{s}_{\mathcal{E}} Q$

SRC-UPDATE-MONO-FUPD
$\Rightarrow\!\!\!\!\!\!\Rightarrow^{s}_{\mathcal{E}} P * (P \Rrightarrow\!\!\!\!\!\!\Rightarrow_{\mathcal{E}} Q) \vdash \Rightarrow\!\!\!\!\!\!\Rightarrow^{s}_{\mathcal{E}} Q$

FUPD-SRC-UPDATE
$\Rightarrow\!\!\!\!\Rightarrow_{\mathcal{E}} \Rightarrow\!\!\!\!\!\!\Rightarrow^{s}_{\mathcal{E}} P \vdash \Rightarrow\!\!\!\!\!\!\Rightarrow^{s}_{\mathcal{E}} P$

Trivial source language The trivial source language is given by the unit type $\mathbb{1}$ with the trivial relation $x \hookrightarrow x \triangleq \top$ and the interpretation $I(x) \triangleq \top$. Essentially, this source language always loops, so that a refinement with this source language poses no further requirements.

Authoritative source language

Definition 7 (Authoritative source)**.** An authoritative source $(M, \hookrightarrow)$ consists of a discrete unital camera M equipped with a relation $\hookrightarrow$ satisfying the following requirements:

$$a \hookrightarrow a' \land \mathsf{valid}(a \cdot f) \Rightarrow \mathsf{valid}(a' \cdot f) \land (a \cdot f) \hookrightarrow (a' \cdot f) \qquad \text{(AUTH-SOURCE-FRAME)}$$
$$\mathsf{valid}(a \cdot f) \land a \cdot f = a \cdot f' \Rightarrow f = f' \qquad \text{(AUTH-SOURCE-CANCEL)}$$

We can make an authoritative source $(M, \hookrightarrow)$ into a source language by defining the source interpretation as

$$I(a) \triangleq \boxed{\bullet\, a}^{\gamma}. \qquad \text{(AUTH-SOURCE-INTERP)}$$

for some ghost name γ. We use the notation $\mathsf{srcA}(s) \triangleq \boxed{\bullet\, a}^{\gamma}$ to assert authoritative ownership of the source element s and $\mathsf{srcF}(s) \triangleq \boxed{\circ\, a}^{\gamma}$ to assert ownership of a fragment s.

The following rules can be proved:

AUTH-SOURCE-UPDATE
$$\frac{a \hookrightarrow a'}{\mathsf{srcF}(a) \vdash \Rightarrow\!\!\!\!\!\!\Rightarrow^{s}_{\mathcal{E}} \mathsf{srcF}(a')}$$

SRCF-SPLIT
$\mathsf{srcF}(a \cdot b) \dashv\vdash \mathsf{srcF}(a) * \mathsf{srcF}(b)$

Authoritative ordinal source language We can give an instance of the authoritative source language by picking the ordinal camera from Section 3.5 as the underlying discrete unital camera. As for the stepping relation, we pick $\succ$ (the greater-than relation).

Intuitively, if we have $\mathsf{srcF}(a)$, then especially $a \preceq b$ for the authoritative element b, asserting that at least a is remaining in the ordinal source (formally, whenever we have knowledge of $\mathsf{srcA}(b)$, then $a \preceq b$).

Naturally, we can derive the following rule (recalling the definition of the operation for the ordinal RA):

ORD-SRCF-SPLIT
$$\mathsf{srcF}(n \oplus m) \dashv\vdash \mathsf{srcF}(n) * \mathsf{srcF}(m)$$

Authoritative natural numbers source language In a very similar way, natural numbers can be used as a source language, with the stepping relation $\hookrightarrow \triangleq >$. The observations for the ordinal source language hold here as well: if we have knowledge of $\mathsf{srcF}(n)$, then we have at least n steps remaining in the natural numbers source. A similar splitting rule holds:

NAT-SRCF-SPLIT
$$\mathsf{srcF}(n + m) \dashv\vdash \mathsf{srcF}(n) * \mathsf{srcF}(m)$$

Lexicographic source language Given two source languages $(A, \hookrightarrow_A, I_A)$ and $(B, \hookrightarrow_B, I_B)$, we can define a source language on the lexicographic product as follows:

$$\frac{a \hookrightarrow_A a'}{(a, b) \hookrightarrow_{A \times B} (a', b')} \qquad \frac{b \hookrightarrow_B b'}{(a, b) \hookrightarrow_{A \times B} (a, b')}$$

$$I(a, b) \triangleq I(a) * I(b)$$

This will allow us to encode stuttering explicitly.

SOURCE-UPDATE-EMBED-L
$$\overset{s}{\Rrightarrow}_{\mathcal{E}}{}^{A}\, P \vdash \overset{s}{\Rrightarrow}_{\mathcal{E}}{}^{(A \times B)}\, P$$

SOURCE-UPDATE-EMBED-R
$$\overset{s}{\Rrightarrow}_{\mathcal{E}}{}^{B}\, P \vdash \overset{s}{\Rrightarrow}_{\mathcal{E}}{}^{(A \times B)}\, P$$

Other source languages In the case that we want to prove a refinement between programs, the source language needs to capture the language the source program is written in. We will see an example of such an encoding in Section 11.2.

9.3.2 Definition of the refinement weakest precondition

We parameterize the refinement weakest precondition by a state interpretation $\mathit{State} \times \mathbb{N} \to \mathit{iProp}$ that interprets the execution state as an Iris proposition, and a predicate $\Phi_F : \mathit{Val} \to \mathit{iProp}$ serving as a postcondition for forked-off threads[24]. For the state interpretation, we currently do not support lists of observations for prophecies (unlike the Iris weakest precondition for safety). Furthermore, we parameterize over a source language $\mathcal{A} = (A, \hookrightarrow, I)$ for refinement which we have to thread through.

The key difference to the normal weakest precondition of Iris is that we have to build in making steps of the source language: we should always be required to take a step in the source language after finitely many steps in the target language. This ensures that, if the source expression is terminating (has no infinite

[24] However, we do not actually make use of concurrency in this work.

executions), the target expression (we are considering in the weakest precondition) must also be terminating. Therefore, we define the weakest precondition as a least fixpoint.

At the same time, our definition still has a guardedly recursive component: for every step of the target, we decrease the step-index (encoded by a later), which is needed for the higher-order reasoning of Iris, *i.e.,* to strip laters when opening invariants. Notably, we do not get a later if we do not take a step in the source language. Otherwise, we could still loop infinitely in the target and never take a step in the source by decreasing the step-index (with the later) until it runs out (*i.e.,* becomes zero).

The most important features are highlighted in blue.

$$
\begin{aligned}
&\mathit{rwp}(\mathcal{A}, S, \Phi_F, s) \triangleq \mathsf{least\_fp}\ \mathit{rwp\_rec}.\lambda\, \mathcal{E}, e, \Phi.\\
&\qquad (\exists v.\, \mathrm{expr\_to\_val}(e) = v \land \forall \sigma, n, a.\, I(a) * S(\sigma, n) \Rrightarrow_{\mathcal{E}} I(a) * S(\sigma, n) * \Phi(v)) \lor\\
&\qquad \Big(\mathrm{expr\_to\_val}(e) = \bot \land \forall \sigma, n, a.\, I(a) * S(\sigma, n) \ {}^{\mathcal{E}}\!\Rrightarrow^{\emptyset}\\
&\qquad\quad \exists b.\, {}^{b}\triangleright \Rrightarrow_{\emptyset} (s = \mathsf{NotStuck} \Rightarrow \mathrm{red}(e, \sigma)) * \forall e', \sigma', \vec{e}, \vec{\kappa}.\, (e, \sigma \xrightarrow{\vec{\kappa}}_{\mathsf{t}} e', \sigma', \vec{e}) \ {}^{\emptyset}\!\Rrightarrow^{\mathcal{E}}\\
&\qquad\qquad ((b = \mathsf{true} * \exists a'.\, a \hookrightarrow^{+} a' * I(a')) \lor (b = \mathsf{false} * I(a))) *\\
&\qquad\qquad S(\sigma', n + |\vec{e}|) * \mathit{rwp\_rec}(\mathcal{E}, e', \Phi) * \textstyle\bigstar_{e'' \in \vec{e}}\, \mathit{rwp\_rec}(\top, e'', \Phi_F)\Big)
\end{aligned}
$$

$$
\mathsf{rwp}_{s;\mathcal{E}}^{\mathcal{A};S;\Phi_F}\ e\ \{v.\, P\} \triangleq \mathit{rwp}(\mathcal{A}, S, \Phi_F, s)(\mathcal{E}, e, \lambda\, v.\, P)
$$

When proving a refinement weakest precondition and the target expression is not a value, we can make a choice: either we want to take a step in the source language ($b = \mathsf{true}$) or we do not. In case that we do, we obtain a later in the goal upfront[25]. Otherwise, we cannot obtain a later in the goal. After taking a step in the target language, we have to prove that we can take at least one step in the source language if we chose to do so, while making sure that the source interpretation is upheld.

This definition of the refinement weakest precondition allows us to take stuttering steps both in the source and the target without explicitly counting steps[26].

We remark that this definition becomes quite similar (apart from differences in placement of later) to the "normal" weakest precondition for safety known from Iris if we pick the trivial source language that can always do a step (thus putting no obligation for termination on us).

We provide additionally a *strong refinement weakest precondition* which captures the case that we are not in the value case and have potentially already taken a source step (made a source update). It requires us to prove a target step. The strong refinement weakest precondition is parameterized by an index k giving the number of laters we may get in the goal when proving it. The interesting cases in this work are $k = 0$ (we have not taken a source step when switching from the refinement weakest precondition to the strong version) and $k = 1$ (we have taken a source step).

$$
\begin{aligned}
\mathsf{rswp}_{s;\mathcal{E}}^{k;\mathcal{A};S;\Phi_F}\ e\ \{v.\, P\} \triangleq\ & \forall \sigma, n, a.\, I(a) * S(\sigma, n) \ {}^{\mathcal{E}}\!\Rrightarrow^{\emptyset} \left({}^{\emptyset}\!\Rrightarrow^{\emptyset} \triangleright\, {}^{\emptyset}\!\Rrightarrow^{\emptyset}\right)^{k}\\
&(s = \mathsf{NotStuck} \Rightarrow \mathrm{red}(e, \sigma)) * \forall e', \sigma', \vec{e}, \vec{\kappa}.\, (e, \sigma \xrightarrow{\vec{\kappa}}_{\mathsf{t}} e', \sigma', \vec{e}) \ {}^{\emptyset}\!\Rrightarrow^{\mathcal{E}}\\
&\quad I(a) * S(\sigma', n + |\vec{e}|) * \mathsf{rwp}_{s;\mathcal{E}}^{\mathcal{A};S;\Phi_F}\ e'\ \{v.\, P\} * \textstyle\bigstar_{e'' \in \vec{e}}\, \mathsf{rwp}_{s;\top}^{\mathcal{A};S;\Phi_F}\ e''\ \{\Phi_F\}
\end{aligned}
$$

The style of reasoning this allows will become clearer with the rules provided below.

[25] In principle, we could also get a larger number of laters here, *e.g.,* by including a logical step. We leave this for future work.

[26] However, we cannot obtain laters in the goal when proving a refinement weakest precondition and stuttering in the target.

9.3.3 Derived rules

We first show the two central rules that allow for reasoning involving the source language.

RWP-NO-STEP
$$\frac{\text{expr\_to\_val}(e) = \bot}{\mathsf{rswp}^{0}_{s;\mathcal{E}}\, e\, \{v.\, P\} \vdash \mathsf{rwp}_{s;\mathcal{E}}\, e\, \{v.\, P\}}$$

RWP-TAKE-STEP
$$\frac{\text{expr\_to\_val}(e) = \bot}{\Rrightarrow^{s}_{\mathcal{E}}\, P * (P \mathrel{-\!\!*} \mathsf{rswp}^{1}_{s;\mathcal{E}}\, e\, \{v.\, P\}) \vdash \mathsf{rwp}_{s;\mathcal{E}}\, e\, \{v.\, P\}}$$

The first rule does not take a step in the source language, thus leaving us at an rswp with 0 laters, while the second rule takes a source step.

Rules for rwp The other rules we expect of weakest preconditions do also hold.

RWP-VALUE
$$P[v/x] \vdash \mathsf{rwp}_{s;\mathcal{E}}\, v\, \{x.\, P\}$$

RWP-MONO
$$\frac{\mathcal{E}_1 \subseteq \mathcal{E}_2 \qquad \Gamma, x : \mathsf{val} \mid P \vdash \Rrightarrow_{\mathcal{E}_2} Q \qquad (s_2 = \mathsf{Stuck} \vee s_1 = s_2)}{\Gamma \mid \mathsf{rwp}_{s_1;\mathcal{E}_1}\, e\, \{x.\, P\} \vdash \mathsf{rwp}_{s_2;\mathcal{E}_2}\, e\, \{x.\, Q\}}$$

FUP-RWP
$$\Rrightarrow_{\mathcal{E}} \mathsf{rwp}_{s;\mathcal{E}}\, e\, \{x.\, P\} \vdash \mathsf{rwp}_{s;\mathcal{E}}\, e\, \{x.\, P\}$$

RWP-FUP
$$\mathsf{rwp}_{s;\mathcal{E}}\, e\, \{x.\, \Rrightarrow_{\mathcal{E}} P\} \vdash \mathsf{rwp}_{s;\mathcal{E}}\, e\, \{x.\, P\}$$

RWP-ATOMIC
$$\frac{\text{strongly\_atomic}(e)}{{}^{\mathcal{E}_1}\Rrightarrow^{\mathcal{E}_2} \mathsf{rwp}_{s;\mathcal{E}_2}\, e\, \left\{x.\ {}^{\mathcal{E}_2}\Rrightarrow^{\mathcal{E}_1} P\right\} \vdash \mathsf{rwp}_{s;\mathcal{E}_1}\, e\, \{x.\, P\}}$$

RWP-FRAME
$$Q * \mathsf{rwp}_{s;\mathcal{E}}\, e\, \{x.\, P\} \vdash \mathsf{rwp}_{s;\mathcal{E}}\, e\, \{x.\, Q * P\}$$

RWP-BIND
$$\frac{K \text{ is a context}}{\mathsf{rwp}_{s;\mathcal{E}}\, e\, \left\{x.\, \mathsf{rwp}_{s;\mathcal{E}}\, K(\text{val\_to\_expr}(x))\, \{y.\, P\}\right\} \vdash \mathsf{rwp}_{s;\mathcal{E}}\, K(e)\, \{y.\, P\}}$$

Finally, we have the usual lifting lemma:

RWP-LIFT-STEP
$$\frac{\text{expr\_to\_val}(e) = \bot}{\begin{array}{l}\forall \sigma, n, a.\, I(a) * S(\sigma, n) \ {}^{\mathcal{E}}\!\Rrightarrow\!\!\!\!\!\!*^{\emptyset} \\ \quad \exists b.\ {}^{b}\triangleright \Rrightarrow_{\emptyset} (s = \mathsf{NotStuck} \Rightarrow \mathrm{red}(e, \sigma)) * \forall e', \sigma', \vec{e}, \vec{\kappa}.\, (e, \sigma \xrightarrow{\vec{\kappa}}_{\mathsf{t}} e', \sigma', \vec{e}) \ {}^{\emptyset}\!\Rrightarrow\!\!\!\!\!\!*^{\mathcal{E}} \\ \qquad ((b = \mathsf{true} \wedge \exists a'.\, a \hookrightarrow^{+} a' * I(a')) \vee (b = \mathsf{false} \wedge I(a))) * \\ \qquad S(\sigma', n + |\vec{e}|) * \mathsf{rwp}^{\mathcal{A};S;\Phi_F}_{s;\mathcal{E}}\, e'\, \{x.\, P\} * \ast_{e'' \in \vec{e}}\, \mathsf{rwp}^{\mathcal{A};S;\Phi_F}_{s;\mathcal{E}}\, e''\, \{\Phi_F\} \\ \vdash \mathsf{rwp}^{\mathcal{A};S;\Phi_F}_{s;\mathcal{E}}\, e\, \{x.\, P\}\end{array}}$$

Rules for rswp Correspondingly, we have the following rules for the rswp:

RSWP-DO-STEP
$$\triangleright\ \mathsf{rswp}^{k}_{s;\mathcal{E}}\, e\, \{x.\, P\} \vdash \mathsf{rswp}^{(1+k)}_{s;\mathcal{E}}\, e\, \{x.\, P\}$$

RSWP-MONO
$$\frac{\mathcal{E}_1 \subseteq \mathcal{E}_2 \qquad \Gamma, x : \mathsf{val} \mid P \vdash \Rrightarrow_{\mathcal{E}_2} Q \qquad (s_2 = \mathsf{Stuck} \vee s_1 = s_2)}{\Gamma \mid \mathsf{rswp}^k_{s_1;\mathcal{E}_1}\, e\, \{x.\, P\} \vdash \mathsf{rswp}^k_{s_2;\mathcal{E}_2}\, e\, \{x.\, Q\}}$$

FUP-RSWP
$$\Rrightarrow_{\mathcal{E}} \mathsf{rswp}^k_{s;\mathcal{E}}\, e\, \{x.\, P\} \vdash \mathsf{rswp}^k_{s;\mathcal{E}}\, e\, \{x.\, P\}$$

RSWP-FUP
$$\mathsf{rswp}^k_{s;\mathcal{E}}\, e\, \{x.\, \Rrightarrow_{\mathcal{E}} P\} \vdash \mathsf{rswp}^k_{s;\mathcal{E}}\, e\, \{x.\, P\}$$

RSWP-ATOMIC
$$\frac{\mathrm{strongly\_atomic}(e)}{{}^{\mathcal{E}_1}\Rrightarrow^{\mathcal{E}_2} \mathsf{rswp}^k_{s;\mathcal{E}_2}\, e \left\{x.\, {}^{\mathcal{E}_2}\Rrightarrow^{\mathcal{E}_1} P\right\} \vdash \mathsf{rswp}^k_{s;\mathcal{E}_1}\, e\, \{x.\, P\}}$$

RSWP-FRAME
$$Q * \mathsf{rswp}^k_{s;\mathcal{E}}\, e\, \{x.\, P\} \vdash \mathsf{rswp}^k_{s;\mathcal{E}}\, e\, \{x.\, Q * P\}$$

RSWP-BIND
$$\frac{K \text{ is a context}}{\mathsf{rswp}^k_{s;\mathcal{E}}\, e \left\{x.\, \mathsf{rwp}_{s;\mathcal{E}}\, K(\mathrm{val\_to\_expr}(x))\, \{y.\, P\}\right\} \vdash \mathsf{rswp}^k_{s;\mathcal{E}}\, K(e)\, \{y.\, P\}}$$

RSWP-LIFT-STEP
$$\frac{\mathrm{expr\_to\_val}(e) = \bot}{\begin{array}{l}\forall \sigma, n.\, S(\sigma, n) \mathrel{-\!\!*} {}^{\mathcal{E}}\!\Longrightarrow^{\emptyset}_{k} \\ \quad (s = \mathsf{NotStuck} \Rightarrow \mathrm{red}(e, \sigma)) * \forall e', \sigma', \vec{e}, \vec{\kappa}.\, (e, \sigma \xrightarrow{\vec{\kappa}}_{\mathsf{t}} e', \sigma', \vec{e}) \Rightarrow {}^{\emptyset}\!\Longrightarrow^{\mathcal{E}} \\ \qquad S(\sigma', n + |\vec{e}|) * \mathsf{rwp}^{\mathcal{A};S;\Phi_F}_{s;\mathcal{E}}\, e'\, \{v.\, P\} * \ast_{e'' \in \vec{e}}\, \mathsf{rwp}^{\mathcal{A};S;\Phi_F}_{s;\top}\, e''\, \{\Phi_F\} \\ \vdash \mathsf{rswp}^{k;\mathcal{A};S;\Phi_F}_{s;\mathcal{E}}\, e\, \{x.\, P\}\end{array}}$$

9.3.4 Adequacy

Of course, we have to prove that our refinement weakest precondition is *adequate*, *i.e.,* that the statement we have proved has a meaning *outside* the logic. When we prove a refinement, we essentially desire that every possible behavior of the target term is also a possible behavior of the source term.

Viewed abstractly with our very general notion of source language in mind, the only behavior we can reason about in general is (non-) termination. More advanced additional notions of behavior, for instance a notion of *result refinement* when the source term is itself a program, can be encoded via the source interpretation with ghost state.

Result refinement Since the concrete phrasing of a result refinement strongly depends on the language and the relation between values we want to obtain, we introduce a generic post-condition depending on the state interpretation and source interpretation. With this, we can prove a general result that can be re-used for proving concrete result refinements.

We make use of the notion of satisfiability with masks introduced in Section 7.5.

Theorem 4 (Result Refinement)**.** *Assume that the underlying step-index type validates the small existential property.*

If $[e]; \sigma \to_{\mathsf{tp}} v :: T; \sigma'$, i.e., *the main thread terminates with a value* v *and state* σ'*, and*

$$\textit{satisfiable\_at}\ \top\ (I(a) * S(\sigma, n) * \textit{rwp}_{s;\top}\, e\, \{\Phi\}),$$

then there exists a source expression a' and a number of threads m such that $a \hookrightarrow^* a'$ and

$$\textit{satisfiable\_at}\ \top\ (I(a') * S(\sigma', m) * \Phi(v)).$$

We will see examples of the use of this general result in Section 11.

Termination refinement One of the strong features that Transfinite Iris offers is that we can not only prove a result refinement, but also that termination is preserved from the source expression to the target expression.

We phrase this as a generic adequacy statement essentially stating that if we can prove a refinement (phrased in terms of the weakest precondition) between a source term s and a target term t and every execution of s terminates, then also every execution of t terminates. Here we take the viewpoint that divergence of the target term is an observable behavior.

To facilitate the proof and for formally defining our adequacy statement, we first lift the essential parts of the rwp relevant for termination-preserving refinement to thread pools, *rwp_tp*. However, while our definition of the refinement weakest precondition does in principle support concurrency, our adequacy statement (and refinement weakest precondition) does not handle fairness. Many interesting, termination-preserving refinement in concurrent settings rely on fairness (*i.e.,* some form of fair scheduling between threads). We leave the study of this for future work.

$$\begin{aligned}
&\textit{rwp_tp} \triangleq \mathsf{least_fp}\ \textit{rwptp_rec}.\lambda\, T.\\
&\quad \forall T', \sigma, \sigma', \vec{\kappa}, n, a.\ \ T; \sigma \to_{\mathsf{tp}} T'; \sigma' \mathrel{-\!\!*}\\
&\qquad \Big(I(a) * S(\sigma, n)\ {}^{\top}\!\Rrightarrow\!\!\!\!\!*^{\emptyset}\ \exists b.\ \overset{b}{\triangleright}\,{}^{\emptyset}\!\Rrightarrow^{\top} \exists m.\ S(\sigma', m) *\\
&\qquad ((b = \mathsf{true} * \exists a'.\ a \hookrightarrow^+ a' * I(a')) \vee (b = \mathsf{false} * I(a))) * \textit{rwptp_rec}\, T'\Big)
\end{aligned}$$

Essentially, this removes the progress requirement, the post condition, as well as the separate handling of the value case, leaving us with what is essential for dealing with simulations.

Importantly, we can prove the following subsumption for single threads:

$$\begin{array}{l}\textsc{rwp-rwp\_tp}\\ \mathsf{rwp}_{s;\top}\ e\ \{v.\ P\} \vdash \textit{rwp\_tp}[e]\end{array}$$

Now for proving adequacy we employ the satisfiability technique from Section 4.4.1. We first define a predicate $\textit{guarded} : A \to \textit{iProp} \to \textit{iProp}$ which essentially captures that a proposition P is guarded by an element a of the source language. If we can evaluate a long enough that it terminates (cannot be reduced anymore by $\hookrightarrow$), then guardedness ensures that we get knowledge of P.

$$\begin{aligned}
\textit{guarded} \triangleq \mu\ \ &\textit{guarded_pre}.\ \lambda\,(a : A)(P : \textit{iProp}).\ {}^{\top}\!\Rrightarrow^{\emptyset}\\
&({}^{\emptyset}\!\Rrightarrow^{\top} P) \vee (\triangleright\,{}^{\emptyset}\!\Rrightarrow^{\top} \exists a'.\ a \hookrightarrow^+ a' * \textit{guarded_pre}.a' P)
\end{aligned}$$

We can formalize the intuition for guardedness if we assume a step-index type validating the existential property. Intuitively, if we have a terminating execution of the source language, we can just pick a step-index which is larger than the execution length to be able to trace the whole execution and in the end obtain P.

Formally, we define strong normalisation (at the meta-level) as follows:

$$\frac{\forall y. R\ x\ y \to \mathsf{SN}\ R\ y}{\mathsf{SN}\ R\ x}$$

Moreover, for a relation $R : A \to A \to \mathit{Prop}$, we define co-inductively what it means that there is an infinite path from an element $a : A$:

$$\frac{R\ x\ y \qquad \mathsf{loops}\ R\ y}{\mathsf{loops}\ R\ x}$$

Now we can prove the intuition:

Proposition 1. *Assume that the underlying step-index type validates the small existential property.*

$$\frac{\textit{GUARDED-SAT}\quad \mathit{SN}\ (\hookrightarrow)\ a \qquad \mathit{satisfiable\_at}\ \top\ (\mathit{guarded}\ a\ P)}{\mathit{satisfiable\_at}\ \top\ P}$$

Proof. By induction over the proof of strong normalisation using the compatibility lemmas for satisfiable_at $\mathcal{E}$. □

Theorem 5 (Termination refinement)**.** *Assume that the underlying step-index type validates the small existential property.*

If $\mathit{SN}\ (\hookrightarrow)\ a$ *and* $\mathit{satisfiable\_at}\ \top\ (I(a) * S(\sigma, n) * \mathit{rwp}_{s;\top}\ e\ \{v.\ P\})$ *then* $\mathit{loops}\ (\to_{\mathsf{tp}})([e]; \sigma)$ *is false.*

Essentially, the preceding theorem tells us that if the source is strongly normalising and we can prove the refinement weakest precondition for the target, then also the target is strongly normalizing[27].

9.3.5 Sequential weakest precondition

For sequential reasoning about liveness reasoning (which we focus on in this work), it is useful to have a version of the refinement weakest precondition using the non-atomic invariants of §7.4. This removes the obligation to write our algorithms in a thread-safe way, allowing us to open invariants around non-atomic expressions.

The definition as a derived form of rwp is simple:

$$\mathsf{seqrwp}^{\mathcal{A};S;\Phi_F}_{s;\mathcal{E}}\ e\ \{v.\ P\} \triangleq [\mathsf{NaInv} : \gamma_{\mathsf{seq}}.\mathcal{E}] \mathrel{-\!\!*} \mathsf{rwp}^{\mathcal{A};S;\Phi_F}_{s;\top}\ e\ \big\{v.\ [\mathsf{NaInv} : \gamma_{\mathsf{seq}}.\mathcal{E}] * P\big\},$$

where γ_{seq} is some arbitrary ghost variable name which we fix beforehand.

Note how the mask parameter is just used for the non-atomic invariants, while we pass the full $\top$ mask to the rwp.

In a similar way, a sequential version of the strong refinement weakest precondition can be defined:

$$\mathsf{seqrswp}^{k;\mathcal{A};S;\Phi_F}_{s;\mathcal{E}}\ e\ \{v.\ P\} \triangleq [\mathsf{NaInv} : \gamma_{\mathsf{seq}}.\mathcal{E}] \mathrel{-\!\!*} \mathsf{rswp}^{k;\mathcal{A};S;\Phi_F}_{s;\top}\ e\ \big\{v.\ [\mathsf{NaInv} : \gamma_{\mathsf{seq}}.\mathcal{E}] * P\big\},$$

[27] Strong normalization is classically equivalent to the absence of loops.

9.4 Time-credits Weakest Precondition

We can instantiate the refinement weakest precondition with ordinals as a source language, with the stepping relation given by the usual total order $>$, in order to obtain a *total* weakest precondition that ensures termination. Formally, we make use of the authorative ordinal source language given previously.

This gives us a generalization of time credits Mével et al. [2019]. In their original form, time credits enable proving the safety property *bounded termination* in separation logic. That is, with time credits one can verify (explained below) that a program "terminates in n steps of computation" where the bound n has to be fixed up-front. By picking ordinals, we obtain transfinite time credits. Transfinite time credits go beyond the safety property bounded termination: they allows us to prove the liveness property *termination* for examples where it is non-trivial (if not impossible) to determine sufficient finite bounds.

We use the established notation

$$\$(\alpha) \triangleq \mathsf{srcF}(\alpha)$$

for asserting that there are at least α credits left.

Clearly, we then obtain the rule

TC-SPLIT
$$\$(\alpha \oplus \beta) \dashv\vdash \$(\alpha) * \$(\beta)$$

The *time-credits weakest precondition* is defined by

$$\mathsf{twp}_{s;\mathcal{E}}^{S;\Phi_F}\ e\ \{v.\ P\} \triangleq \mathsf{rwp}_{s;\mathcal{E}}^{\mathsf{Ord};S;\Phi_F}\ e\ \{v.\ P\},$$

just specializing the source language to the ordinal source language.

The following rules are of relevance:

TCWP-BURN-CREDIT
$$\frac{\mathrm{expr\_to\_val}(e) = \bot}{\$1 * \triangleright \mathsf{rswp}_{s;\mathcal{E}}^{0}\ e\ \{v.\ P\} \vdash \mathsf{twp}_{s;\mathcal{E}}\ e\ \{v.\ P\}}$$

TCWP-WEAKEN
$$\frac{\mathrm{expr\_to\_val}(e) = \bot \qquad \beta \preceq \alpha}{(\$\beta \mathrel{-\!\!*} \mathsf{twp}_{s;\mathcal{E}}\ e\ \{v.\ P\}) * \$\alpha \vdash \mathsf{twp}_{s;\mathcal{E}}\ e\ \{v.\ P\}}$$

TCWP-ALLOC-ZERO
$$\$0 * \mathsf{twp}_{s;\mathcal{E}}\ e\ \{v.\ P\} \vdash \mathsf{twp}_{s;\mathcal{E}}\ e\ \{v.\ P\}$$

Adequacy Since the total order on ordinals is well-founded, every ordinal is strongly normalising according to the $\succ$ relation. Thus we can instantiate the generic termination refinement result (Theorem 5).

Theorem 6 (Adequacy of time-credits)**.** *Assume that the underlying step-index type validates the small existential property and that*

$$\mathit{satisfiable\_at}\ \top\ (\mathit{srcA}(\alpha) * S(\sigma)(n) * \mathit{twp}_{s;\top}\ e\ \{v.\ P\}).$$

Then $\mathit{loops}(\rightarrow_{\mathsf{tp}})([e];\sigma)$ *is false, which is classically equivalent to* e *being strongly normalising.*

Sequential version Similarly to the refinement weakest precondition, we can define a sequential variant of the time-credits weakest precondition:

$$\mathsf{seqtwp}_{s;\mathcal{E}}^{S;\Phi_F}\ e\ \{v.\ P\} \triangleq [\mathrm{NaInv} : \gamma_{\mathsf{seq}}.\mathcal{E}] \mathrel{-\!\!*} \mathsf{twp}_{s;\top}^{S;\Phi_F}\ e\ \big\{v.\ [\mathrm{NaInv} : \gamma_{\mathsf{seq}}.\mathcal{E}] * P\big\},$$

9.5 Hoare Triples

While reasoning directly with weakest preconditions is in general easier (in particular in Coq), a more traditional way of writing down specifications are Hoare Triples. We can easily define Hoare Triples in terms of the refinement weakest precondition and strong refinement weakest precondition:

$$\{P\}\, e\, \{v.\, Q\}_{s;\mathcal{E}} \triangleq \Box (P \twoheadrightarrow \mathsf{rwp}_{s;\mathcal{E}}\, e\, \{Q\})$$
$$\langle P\rangle\, e\, \langle v.\, Q\rangle^{k}_{s;\mathcal{E}} \triangleq \Box (P \twoheadrightarrow \mathsf{rswp}^{k}_{s;\mathcal{E}}\, e\, \{Q\})$$
$$\langle P\rangle\, e\, \langle v.\, Q\rangle_{s;\mathcal{E}} \triangleq \Box (P \twoheadrightarrow \mathsf{rswp}^{0}_{s;\mathcal{E}}\, e\, \{Q\})$$

We will overload these notations for the sequential versions. It will always be clear with which weakest precondition we are working.

9.5.1 General Hoare Triples for the Refinement Weakest Precondition

The following rules can be derived for the refinement weakest precondition:

RHT-VALUE
$$\{\mathsf{True}\}\, v\, \{w.\, v = w\}_{\mathcal{E}}$$

RHT-FRAME
$$\frac{\{P\}\, e\, \{v.\, Q\}_{\mathcal{E}}}{\{P * R\}\, e\, \{v.\, Q * R\}_{\mathcal{E}}}$$

RHT-BIND
$$\frac{\{P\}\, e\, \{v.\, Q\}_{\mathcal{E}} \qquad \forall v.\, \{Q\}\, K[v]\, \{w.\, R\}_{\mathcal{E}}}{\{P\}\, K[e]\, \{w.\, R\}_{\mathcal{E}}}$$

$$
\begin{aligned}
v : \mathsf{Val} &\triangleq i \in \mathbb{Z} \mid () \mid l \in \mathsf{Loc} \mid \textbf{true} \mid \textbf{false} \mid \mathsf{inl}(v) \mid \mathsf{inr}(v) \mid (v_1, v_2) \mid \mathsf{rec}\ f\ x.e \\
e : \mathsf{Expr} &\triangleq x \mid v \mid \mathsf{inl}(e) \mid \mathsf{inr}(e) \mid (e_1, e_2) \mid \pi_i(e) \mid \mathsf{match}\ e\ \mathsf{with}\ \mathsf{inl}(x).e_1 \mid \mathsf{inr}(x).e_2 \\
&\quad \mid \mathsf{if}\ e\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 \mid e_1(e_2) \mid \mathsf{ref}(e) \mid \mathbf{!}e \mid e_1 := e_2 \mid e_1\ o\ e_2 \mid u\ e \\
o : \mathsf{BinOp} &\triangleq + \mid - \mid \cdot \mid \leq \mid = \mid \&\& \mid \ldots \\
u : \mathsf{UnOp} &\triangleq - \mid \sim \\
K : \mathsf{Ctx} &\triangleq \bullet \mid \mathsf{inl}(K) \mid \mathsf{inr}(K) \mid (K, e) \mid (v, K) \mid \mathbf{!}K \mid K\ o\ e \mid v\ o\ K \mid \ldots \\
\sigma : \mathsf{State} &\triangleq \mathsf{Loc} \xrightarrow{\text{fin}} \mathsf{Val}
\end{aligned}
$$

Figure 3: Syntax of HeapLang, evaluation contexts, and states.

10 HeapLang

The standard language that is shipped with Iris is *HeapLang*. *HeapLang* is a call-by-value λ-calculus with recursive functions, general higher-order references and fork-style concurrency. Integers and Booleans are built-in as primitives. HeapLang is *a priori* untyped with no fixed static typing judgment. Typing relations (and in this way also polymorphic types), however, can be defined on top depending on the application (we will see an example in §12).

Figure 3 shows the syntax of HeapLang as well as an excerpt of the definition of evaluation contexts. We omit some parts of the language, in particular the support for prophecies and some concurrency primitives, as they are not relevant in this work.

Some syntacic sugar can be defined on top:

$$
\begin{aligned}
\lambda\ x.e &\triangleq \mathsf{rec}\ \_\ x.e \\
\mathsf{let}\ x := e\ \mathsf{in}\ e' &\triangleq (\lambda\ x.e')(e) \\
\mathsf{let}\ (x, y) := e\ \mathsf{in}\ e' &\triangleq (\lambda\ p.\lambda\ f.f(\pi_1 p)(\pi_2 p))(e)(\lambda\ x.\lambda\ y.e')
\end{aligned}
$$

We define a primitive (thread-local) reduction relation in Figure 4, using the same notation for thread-local reductions $\rightarrow_{\mathsf{t}}$ as in the previous section on general languages. This is completely standard. HeapLang defined in this way fulfills the definition of a *language* according to the previous section. Thus, we can make use of the general definition for machine reduction.

10.1 Heap Encoding and State Interpretation

When we want to reason about HeapLang using the Iris program logic, we have to define its resources as ghost state and provide a state interpretation to use for the weakest precondition. We simplify the presentation here and omit HeapLang's support for prophecies as they are irrelevant for the work at hand[28]. More specifically, in the following we define the points-to connective $l \mapsto v$ and the points-to connective with fractional permissions $l \overset{q}{\mapsto} v$.

[28] Iris' actual generic support for heaps is even more complicated, allowing to track additional meta information about each location. This is of no relevance to us and therefore omitted.

ADD
$$\frac{v_1 = n_1 \qquad v_2 = n_2}{v_1 + v_2 \to_{\mathsf{pure}} (n_1 + n_2)}$$

PROJECT
$$\pi_i(v_1, v_2) \to_{\mathsf{pure}} v_i$$

MATCH-RIGHT
$$(\mathsf{match}\ \mathsf{inl}(v)\ \mathsf{with}\ \mathsf{inl}(x).e_1 \mid \mathsf{inr}(x).e_2) \to_{\mathsf{pure}} e_1[v/x]$$

MATCH-LEFT
$$(\mathsf{match}\ \mathsf{inr}(v)\ \mathsf{with}\ \mathsf{inl}(x).e_1 \mid \mathsf{inr}(x).e_2) \to_{\mathsf{pure}} e_2[v/x]$$

BETA
$$(\mathsf{rec}\ f\ x.e)v \to_{\mathsf{pure}} e[v/x][\mathsf{rec}\ f\ x.e/f]$$

PURE
$$\frac{e_1 \to_{\mathsf{pure}} e_2}{e_1, \sigma \to_{\mathsf{head}} e_2, \sigma}$$

DEREF
$$\frac{\sigma(l) = v}{!l, \sigma \to_{\mathsf{head}} v, \sigma}$$

STORE
$$\frac{\sigma(l) = v_0}{l := v, \sigma \to_{\mathsf{head}} (), \sigma[l \leftarrow v]}$$

ALLOC
$$\frac{\sigma(l) = \bot}{\mathsf{ref}(v), \sigma \to_{\mathsf{head}} l, \sigma[l \leftarrow v]}$$

FORK
$$\mathsf{fork}\{e\}, \sigma \to_{\mathsf{head}} (), \sigma, [e]$$

HEAD
$$\frac{e, \sigma \to_{\mathsf{head}} e, \sigma, \vec{e}}{K(e), \sigma \to_{\mathsf{t}} K(e), \sigma, \vec{e}}$$

Figure 4: Selected rules for pure ($\to_{\mathsf{pure}}$) reduction, head reduction ($\to_{\mathsf{head}}$) and primitive (thread-local) ($\to_{\mathsf{t}}$) reduction in HeapLang.

The resources of HeapLang, namely its heap, are encoded, just as other resources, using a camera. The heap supports fractional permissions. Specifically, we define the following unital camera, where we are equipping Val with a discrete camera structure:

$$\mathsf{heap}_0 \triangleq \mathsf{Loc} \xrightarrow{\mathsf{fin}} (\textsc{Frac} \times \textsc{Ag}(\mathsf{Val}))$$
$$\mathsf{heap} \triangleq \textsc{Auth}(\mathsf{heap}_0)$$

This camera keeps track of the values in individual locations. The interaction of the fractional camera and the agreement camera are quite important. Especially in the case where the fraction is 1 (*i.e.,* we should intuitively have full ownership of a heap location), the product of the two has an exclusive behavior, as fractions cannot be 0. In the absence of fractions, it may thus be more intuitive to imagine the definition as $\mathsf{Loc} \xrightarrow{\mathsf{fin}} \textsc{Ex}(\mathsf{Val})$.

The authoritative camera at the outside ensures that we can hand out fragments of the heap (enabling shared access to the heap), while there will always be exactly one authoritative heap. This authoritative heap is managed by the state interpretation of the weakest precondition and is tied to the heap used for physical reduction. We can canonically turn a physical heap $\sigma : l \xrightarrow{\mathsf{fin}} \mathsf{Val}$ into an element of heap_0 by (pointwise) using the injection ag into the agreement camera and using the fraction 1. This injection will be used implicitly whereever necessary from now on.

Let us pick a ghost name γ_{heap} for the heap. We can now define the points-to connectives using fragmental ownership:

$$l \overset{q}{\mapsto} v \triangleq \boxed{\circ\,([l \leftarrow (q, \mathsf{ag}(v))])}^{\gamma_{\mathsf{heap}}}$$
$$l \mapsto v \triangleq l \overset{1}{\mapsto} v$$

The following rules about the points-to connective can be derived:

POINTSTO-TIMELESS
$\mathsf{timeless}(l \overset{q}{\mapsto} v)$

POINTSTO-COMBINE
$l \overset{q_1}{\mapsto} v_1 * l \overset{q_2}{\mapsto} v_2 \vdash l \overset{q_1+q_2}{\mapsto} v_1 * v_1 = v_2$

HEAP-ALLOC
$$\frac{\sigma(l) = \bot}{\boxed{\bullet\, \sigma}^{\gamma_{\mathsf{heap}}} \vdash \dot{\Rrightarrow}(\boxed{\bullet\, \sigma\, [l \leftarrow v]}^{\gamma_{\mathsf{heap}}} * l \mapsto v)}$$

HEAP-UPDATE
$\boxed{\bullet\, \sigma}^{\gamma_{\mathsf{heap}}} * l \mapsto v_1 \vdash \dot{\Rrightarrow} \boxed{\bullet\, \sigma\, [l \leftarrow v_2]}^{\gamma_{\mathsf{heap}}} * l \mapsto v_2$

HEAP-VALID
$\boxed{\bullet\, \sigma}^{\gamma_{\mathsf{heap}}} * l \overset{q}{\mapsto} v \vdash \sigma(l) = v$

The state interpretation (ignoring prophecies) can be defined as

$$S(\sigma, n) \triangleq \boxed{\bullet\, \sigma}^{\gamma_{\mathsf{heap}}},$$

essentially asserting that the heap encoded in the ghost state matches the physical heap σ.

11 Termination and Refinements for HeapLang

In this section, we provide instantiations of the program logic for proving both termination and termination-preserving refinement properties in HeapLang with Transfinite Iris.

11.1 Termination with HeapLang

The following theorem states the adequacy of the sequential time-credits weakest precondition for HeapLang, derived from the general adequacy in Theorem 6.

Theorem 7 (Adequacy of time-credits for HeapLang)**.** *Assume that we are working in a step-index type validating the existential property. We use Excluded Middle.*

If we can prove $\mathit{True} \vdash \exists \alpha.\$(\alpha) \twoheadrightarrow \mathit{seqtwp}\ e\ \{v.\ \mathit{True}\}$*, then* e *is strongly normalizing,* i.e., $\mathit{SN}\ (\rightarrow_{\mathsf{tp}})\ ([e]; \sigma)$ *for any* σ*.*

11.2 Refinements with HeapLang

Here we consider how to prove refinements between sequential HeapLang programs. As hinted before, we first have to define a suitable source language, according to Definition 5.

11.2.1 HeapLang Source Language

The challenge with setting up a source language for HeapLang is that we have to encode the full state of a HeapLang program using the source interpretation.

We define the following cameras, using the discrete camera on *Expr* and *State*:

$$\begin{aligned}
\mathsf{tpool} &\triangleq \mathbb{N} \xrightarrow{\mathrm{fin}} \mathrm{Ex}(\mathit{Expr}) \\
\mathsf{cfg}_0 &\triangleq \mathsf{tpool} \times \mathsf{heap}_0 \\
\mathsf{cfg} &\triangleq \textsc{Auth}(\mathsf{cfg}_0) \\
\mathsf{trace} &\triangleq \textsc{Auth}\,(\textsc{MList}\,(\mathsf{List}\ \mathit{Expr} \times (\mathit{Loc} \to \mathit{Val})))
\end{aligned}$$

tpool encodes the current execution state for the involved threads. cfg_0 adds the heap as defined in Section 10.1. Finally, the whole encoding cfg of HeapLang configurations in ghost state wraps this in the authoritative camera to enable sharing of fragments. Thread pools T can be canonically injected into the tpool camera and configurations $T; \sigma$ can be embedded in the cfg camera. We will do this implicitly from now on. trace records a trace of an execution, as a list of all intermediate thread pools and states.

We assume that the cfg camera is available at a location γ_{src}, and the trace camera is available at γ_{trc}. The following connectives for asserting particular execution states of threads or locations can be defined:

$$\begin{aligned}
l \overset{q}{\mapsto}_{\mathsf{src}} v &\triangleq \boxed{\circ\,(\bot, [l \leftarrow (q, \mathsf{ag}(v))])}^{\gamma_{\mathsf{src}}} \\
\mathsf{src}(n \to e) &\triangleq \boxed{\circ\,([n \leftarrow \mathsf{ex}(e)], \bot)}^{\gamma_{\mathsf{src}}} \\
\mathsf{src}(e) &\triangleq \mathsf{src}(0 \to e)
\end{aligned}$$

Given a list l of configurations, we define $\mathsf{tracelist}(l)$ to hold if l represents a trace of steps of exeuction:

$$\begin{aligned}\mathsf{tracelist}([]) &\triangleq \mathsf{True}\\ \mathsf{tracelist}([T;\sigma]) &\triangleq \mathsf{True}\\ \mathsf{tracelist}([T_1;\sigma_1] +\!\!+ [T_2;\sigma_2] +\!\!+ l) &\triangleq T;\sigma \hookrightarrow T';\sigma' \land \mathsf{tracelist}([T_2;\sigma_2] +\!\!+ l)\end{aligned}$$

For the definition of the HeapLang source language, we let source steps be reduction steps in HeapLang and define the source interpretation to assert full authoritative ownership of the given configuration, as well as a trace recording execution to that point.

$$\begin{aligned}T;\sigma \hookrightarrow T';\sigma' &\triangleq T;\sigma \hookrightarrow T';\sigma'\\ I(T;\sigma) &\triangleq \boxed{\bullet (T;\sigma)}^{\gamma_{\mathsf{src}}} * \exists l.\, \boxed{\bullet\, \mathsf{mlist}(l +\!\!+ [T;\sigma])}^{\gamma_{\mathsf{trc}}} \land \mathsf{tracelist}(l +\!\!+ [T;\sigma])\end{aligned}$$

11.2.2 Stuttering HeapLang

For refinements we are sometimes in the situation that one step in the source "corresponds" to multiple steps in the target, or vice versa. That is, we would like to have support for stuttering. While our definition of the refinement weakest precondition already has built-in support for stuttering in the source (allowing to take multiple steps in the source for one step of the target) and stuttering in the target, both without explicitly counting steps, we cannot obtain a later in the goal when we do not take a source step. In case we need to stutter in the target and open invariants, we can however still count steps explicitly.

Precisely, we will use the lexicographic source language presented in Section 9.3.1 to combine the HeapLang source language with the natural numbers source language: $\mathcal{A}_{\mathsf{HeapS}} \triangleq \mathcal{A}_{\mathsf{Heap}} \times \mathcal{A}_{\mathbb{N}}$. Everytime we take a step in the source program, we can pick a natural number describing the number of target steps we want to do before the next source step. We will make use of the same time-credits notation $\$k$ as in Section 9.4 to denote that we have a stutter budget of k remaining[29]. Formally, the rules of the lexicographic source language specialized to this setting are as follows:

SOURCE-UPDATE-HEAP
$$\Rrightarrow_{\mathcal{E}}^{s\,A} P \vdash \Rrightarrow_{\mathcal{E}}^{s\,(A\times B)} P$$

SOURCE-UPDATE-EMBED-R
$$\Rrightarrow_{\mathcal{E}}^{s\,B} P \vdash \Rrightarrow_{\mathcal{E}}^{s\,(A\times B)} P$$

[29] The encoding under the hood really is the same, the only difference being that we are now using natural numbers instead of the ordinals.

We can prove the following source updates:

STEP-FRAME
$$\frac{(c_1, n) \hookrightarrow (c_2, m)}{(c_1, n+k) \hookrightarrow (c_2, m+k)}$$

STEPS-ADD-STUTTER
$$\frac{(c_1, n) \hookrightarrow^+ (c_2, m) \qquad c_1 \neq c_2}{(c_1, n) \hookrightarrow^+ (c_2, m+k)}$$

STEP-INV-ALLOC
$$(\mathsf{src}(j \to e_1) \twoheadrightarrow \Rightarrow^s_{\mathcal{E}} P) * (P \twoheadrightarrow \exists e_2.\, \mathsf{src}(j \to e_2) * e_2 \neq e_1) \vdash \mathsf{src}(j \to e_1) \twoheadrightarrow \Rightarrow^s_{\mathcal{E}} (P * \$k)$$

STEP-STUTTER
$$\$(1+k) \vdash \Rightarrow^s_{\mathcal{E}} \$k$$

STEP-PURE
$$\frac{e_1 \to_{\mathsf{pure}} e_2}{\mathsf{src}(j \to e_1) \vdash \Rightarrow^s_{\mathcal{E}} \mathsf{src}(j \to e_2)}$$

STEP-LOAD
$$\mathsf{src}(j \to K[!l]) * l \overset{q}{\mapsto}_{\mathsf{src}} \vdash \Rightarrow^s_{\mathcal{E}} \mathsf{src}(j \to K[v]) * l \overset{q}{\mapsto}_{\mathsf{src}} v$$

STEP-STORE
$$\mathsf{src}(j \to K[l := v]) * l \mapsto_{\mathsf{src}} v \vdash \Rightarrow^s_{\mathcal{E}} \mathsf{src}(j \to K[()]) * l \mapsto_{\mathsf{src}} v$$

STEP-ALLOC
$$\mathsf{src}(j \to K[\mathsf{ref}(v)]) \vdash \Rightarrow^s_{\mathcal{E}} \exists l.\, \mathsf{src}(j \to K[l]) * l \mapsto_{\mathsf{src}} v$$

11.2.3 Adequacy

From the generic adequacy theorems for the refinement weakest precondition (see §9.3.4), we get the following concrete adequacy theorem for the HeapLang refinement.

Theorem 8 (HeapLang refinement adequacy)**.** *Assume that the underlying step-index type validates the small existential property.*

Let a meta-level predicate $\Phi : Val \times Val \to Prop$ be given, relating source and target values. Let $s; \sigma_{src}$ and $t; \sigma$ be the initial source and target configurations, respectively.

Assume that we have proved

$$src(s) \vdash seqrwp_{\top}\ t\ \{v.\ \exists v'.src(v') * \Phi(v, v')\}.$$

Then we have:

1. *Result Refinement: For any execution $[t]; \sigma \to^*_{\mathsf{tp}} [v]; \sigma'$ of the target, there exist v', σ'_{src}, and a list of threads T such that*
$$[s]; \sigma_{src} \to^*_{\mathsf{tp}} v', T; \sigma'_{src}$$
and $\Phi(v, v')$.
2. *Termination Refinement: If the source is strongly normalizing, then the target is strongly normalizing:*
$$SN\ (\to_{\mathsf{tp}})\ ([s]; \sigma_{src}) \Rightarrow SN\ (\to_{\mathsf{tp}})\ ([t]; \sigma)$$

This result relates the proof done in the Transfinite Iris logic to a termination-preserving refinement at the meta-level.

The proof obligation of this theorem requires us to prove that the target term t evaluates to a value v and we can simultaneously advance the source term s such that the source ends up in a value v' related to v by Φ.

11.2.4 Refinement Hoare Triples

In the following, we specialize to the sequential setting and consider the sequential refinement weakest precondition as well as the strong sequential refinement weakest precondition. While we generally prove specifications directly using weakest preconditions, as an example we provide the following Hoare triples for refinement, where we fix the mask to be $\top$[30]:

STORE-TARGET
$$\langle l \mapsto v_1 \rangle\ l := v_2\ \langle w.\ w = () * l \mapsto v_2 \rangle$$

STORE-SOURCE
$$\frac{\langle l \mapsto_{\mathsf{src}} v_2 * \mathsf{src}(K[()]) * P \rangle\ e_t\ \langle v.\ Q \rangle \qquad e_t \text{ is not a value}}{\{l \mapsto_{\mathsf{src}} v_1 * \mathsf{src}(K[l := v_2]) * \triangleright P\}\ e_t\ \{v.\ Q\}}$$

PURE-TARGET
$$\frac{\{P\}\ e_t'\ \{v.\ Q\} \qquad e_t \to_{\mathsf{pure}} e_t'}{\langle P \rangle\ e_t\ \langle v.\ Q \rangle_{\mathcal{E}}}$$

PURE-SOURCE
$$\frac{\langle \mathsf{src}(K[e_s']) * P \rangle\ e_t\ \langle v.\ Q \rangle \qquad e_s \to_{\mathsf{pure}} e_s' \qquad e_t \text{ is not a value}}{\{\mathsf{src}(K[e_s]) * \triangleright P\}\ e_t\ \{v.\ Q\}}$$

STUTTER-TARGET
$$\frac{\langle P \rangle\ e_t\ \langle v.\ Q \rangle \qquad e_t \text{ is not a value}}{\{P\}\ e_t\ \{v.\ Q\}}$$

STUTTER-SOURCE-PURE
$$\frac{\{\mathsf{src}(e_s') * P\}\ e_t\ \{v.\ Q\} \qquad e_s \to_{\mathsf{pure}} e_s' \qquad e_t \text{ is not a value}}{\{\mathsf{src}(e_s) * P\}\ e_t\ \{v.\ Q\}}$$

STUTTER-SOURCE-STORE
$$\frac{\{\mathsf{src}(K[()]) * l \mapsto_{\mathsf{src}} v_2 * P\}\ e_t\ \{v.\ Q\} \qquad e_t \text{ is not a value}}{\{\mathsf{src}(K[\mathsf{ref}(l)v_2]) * P * l \mapsto_{\mathsf{src}} v_1\}\ e_t\ \{v.\ Q\}}$$

[30] one could however also prove the more general rules for arbitrary masks

12 Logical Relation for Termination

In the following, we explain how we obtain the main result of Spies et al. [2021] in Transfinite Iris: termination of a linear language with asynchronous channels, modeling the core of promises in JavaScript. We explain how to encode their language in HeapLang in Section 12.1, we simplify their logical relation in Section 12.2, and we extend it to handle impredicative polymorphism in Section 12.3.

12.1 Language

The language of Spies et al. [2021], λ_{CHAN} is a linear λ-calculus with Booleans, natural numbers, and pairs extended with asynchronous channels. Besides constructs which are already included in HeapLang (*e.g.,* booleans, pairs, and natural numbers), λ_{CHAN} features a primitive operation iter for iteration on natural numbers as well as the chan, put, and get primitives on asynchronous channels. Below, we explain how to encode these constructs into HeapLang.

Channels can be in three states: empty, containing a value (waiting for a get), or containing a continuation (waiting for a put). We encode these states with three constant values E, $\mathsf{V}(v)$, and $\mathsf{C}(v)$ (we can derive these in HeapLang with nested sums), providing for a corresponding case analysis principle

$$\mathsf{case}\ e\ \mathsf{of}\ \mathsf{E} \Rightarrow e_1 \mid \mathsf{V}(v) \Rightarrow e_2 \mid \mathsf{C}(v) \Rightarrow e_3.$$

Channels are then represented by heap locations which contain one of these states. With these features, we can encode the three essential channel operations as follows:

$$\begin{aligned}
\mathsf{chan} &\triangleq \mathsf{let}\ c := \mathsf{ref}(\mathsf{E})\ \mathsf{in}\ (c, c)\\
\mathsf{div} &\triangleq \mathsf{rec}\ f\ x.f(x)\\
\mathsf{get} &\triangleq \lambda p.\mathsf{let}\ c := \pi_1(p)\ \mathsf{in}\\
&\qquad \mathsf{let}\ v := \pi_2(p)\ \mathsf{in}\\
&\qquad \mathsf{case}\ !c\ \mathsf{of}\ \mathsf{E} \Rightarrow c := \mathsf{V}(v) \mid \mathsf{V}(v) \Rightarrow \mathsf{div}() \mid \mathsf{C}(v) \Rightarrow c := \mathsf{E}; f(v)\\
\mathsf{put} &\triangleq \lambda p.\mathsf{let}\ c := \pi_1(p)\ \mathsf{in}\\
&\qquad \mathsf{let}\ f := \pi_2(p)\ \mathsf{in}\\
&\qquad \mathsf{case}\ !c\ \mathsf{of}\ \mathsf{E} \Rightarrow c := \mathsf{C}(f) \mid \mathsf{V}(v) \Rightarrow c := \mathsf{E}; f(v) \mid \mathsf{C}(v) \Rightarrow \mathsf{div}()\\
\mathsf{iter} &\triangleq \mathsf{rec}\ \mathit{iter}\ s.\lambda n, f.\mathsf{if}\ n = 0\ \mathsf{then}\ s\ \mathsf{else}\ \mathit{iter}(f\ s)(n-1)f
\end{aligned}$$

In λ_{CHAN}, there is no case for C in get and V in put. Being interested in termination, we have replaced these cases with divergence, ensuring they are never executed.

Type System In its original (monomorphic) formulation, λ_{CHAN} is equipped with the linear type system presented in Figure 5. We write Γ_1, Γ_2 for the disjoint union of the linear typing contexts Γ_1 and Γ_2.

12.2 Simplified Logical Relation

In Transfinite Iris, we define a simplified logical relation for this language. As a semantic interpretation of types SemType, we pick predicates from values to propositions $T, U : \mathsf{Val} \to \mathit{iProp}$. On these predicates, we define the semantic type formers:

$$x : A \vdash x : A \qquad \frac{\Gamma \vdash e : B}{\Gamma, x : A \vdash e : B} \qquad \emptyset \vdash () : \mathbb{1} \qquad \frac{\Gamma_1 \vdash e_1 : \mathbb{1} \qquad \Gamma_2 \vdash e_2 : A}{\Gamma_1, \Gamma_2 \vdash e_1; e_2 : A} \qquad \Gamma \vdash b : \mathbb{B}$$

$$\frac{\Gamma_1 \vdash e : \mathbb{B} \qquad \Gamma_2 \vdash e_1 : A \qquad \Gamma_2 \vdash e_2 : A}{\Gamma_1, \Gamma_2 \vdash \mathsf{if}\ e\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 : A} \qquad \emptyset \vdash n : \mathbb{N} \qquad \frac{\Gamma_1 \vdash e_1 : \mathbb{N} \qquad \Gamma_2 \vdash e_2 : \mathbb{N}}{\Gamma_1, \Gamma_2 \vdash e_1 + e_2 : \mathbb{N}}$$

$$\frac{\Gamma_1 \vdash e : \mathbb{N} \qquad \Gamma_2 \vdash e_0 : A \qquad x : A \vdash e_S : A}{\Gamma_1, \Gamma_2 \vdash \mathsf{iter}\ e_0\ e\ (\lambda\, x.e_S) \vdash A} \qquad \frac{\Gamma, x : A \vdash e : B}{\Gamma \vdash \lambda\, x.e : A \multimap B}$$

$$\frac{\Gamma_1 \vdash e_1 : A \multimap B \qquad \Gamma_2 \vdash e_2 : A}{\Gamma_1, \Gamma_2 \vdash e_1(e_2) : B} \qquad \frac{\Gamma_1 \vdash e_1 : A \qquad \Gamma_2 \vdash e_2 : B}{\Gamma_1, \Gamma_2 \vdash (e_1, e_2) : A \otimes B}$$

$$\frac{\Gamma_1 \vdash e_1 : A_1 \otimes A_2 \qquad \Gamma_1, x : A_1, y : A_2 \vdash e_2 : B}{\Gamma_1, \Gamma_2 \vdash \mathsf{let}\ (x, y) := e_1\ \mathsf{in}\ e_2 : B} \qquad \emptyset \vdash \mathsf{chan}() : \mathsf{Get}(A) \otimes \mathsf{Put}(A)$$

$$\frac{\Gamma_1 \vdash e_1 : \mathsf{Get}(A) \qquad \Gamma_2 \vdash e_2 : A \multimap \mathbb{1}}{\Gamma_1, \Gamma_2 \vdash \mathsf{get}(e_1, e_2) : \mathbb{1}} \qquad \frac{\Gamma_1 \vdash e_1 : \mathsf{Put}(A) \qquad \Gamma_2 \vdash e_2 : A}{\Gamma_1, \Gamma_2 \vdash \mathsf{put}(e_1, e_2) : \mathbb{1}}$$

Figure 5: The (monomorphic) type system of λ_{CHAN}

$$\begin{aligned}
\mathbb{1}_{\text{sem}} &\triangleq \lambda v.v = () \\
\mathbb{B}_{\text{sem}} &\triangleq \lambda v.\exists b : \mathbb{B}.v = b \\
\mathbb{N}_{\text{sem}} &\triangleq \lambda v.\exists n : \mathbb{N}.v = n \\
T \multimap U &\triangleq \lambda f.(\forall v.\, T(v) \twoheadrightarrow \mathcal{E}[\![U]\!](f\, v)) \\
T \otimes U &\triangleq \lambda v.\exists v_1, v_2.v = (v_1, v_2) * T(v_1) * U(v_2) \\
\mathsf{Get}(T) &\triangleq \lambda v.\exists l, \gamma_{\text{get}}, \gamma_{\text{put}}.\, \$1 * v = l * \mathsf{NaInv}^{\gamma_{\text{seq}}.\mathcal{N}.l}(\mathsf{I}(l, T)) * \boxed{\bullet\,()}^{\gamma_{\text{get}}} \\
\mathsf{Put}(T) &\triangleq \lambda v.\exists l, \gamma_{\text{get}}, \gamma_{\text{put}}.\, \$1 * v = l * \mathsf{NaInv}^{\gamma_{\text{seq}}.\mathcal{N}.l}(\mathsf{I}(l, T)) * \boxed{\bullet\,()}^{\gamma_{\text{put}}} \\
\mathcal{E}[\![-]\!] &: \mathsf{Expr} \to \mathit{iProp} \\
\mathcal{E}[\![T]\!] &\triangleq \lambda e.\mathsf{seqtwp}\ e\ \{v.\, T(v)\}
\end{aligned}$$

For the interpretation of the $\mathsf{Get}(\cdot)$ and $\mathsf{Put}(\cdot)$ types, we maintain an invariant saying that the channel location points to a valid channel state. To every channel, ghost names γ_{get} and γ_{put} are associated. Having $\boxed{\bullet\,()}^{\gamma_{\text{get}}}$ (or $\boxed{\bullet\,()}^{\gamma_{\text{put}}}$) encodes the permission to use the get or put operation, respectively. The authoritative camera and the definition of the invariant ensure for instance that, when we possess $\boxed{\bullet\,()}^{\gamma_{\text{get}}}$, the invariant cannot be in the $\mathsf{C}(f)$ case.

$$\mathsf{I}(l, T) \triangleq l \mapsto \mathsf{E} \vee (\exists v.\, l \mapsto \mathsf{V}(v) * T(v) * \boxed{\bullet\,()}^{\gamma_{\text{put}}}) \vee (\exists f.\, l \mapsto \mathsf{C}(f) * (T \multimap \mathbb{1}_{\text{sem}})(f) * \boxed{\bullet\,()}^{\gamma_{\text{get}}})$$

$$x : T \vDash x : T \qquad \frac{\Gamma \vDash e : U}{\Gamma, x : T \vDash e : U} \qquad \emptyset \vDash () : \mathbb{1}_{\text{sem}} \qquad \frac{\Gamma_1 \vDash e_1 : \mathbb{1}_{\text{sem}} \qquad \Gamma_2 \vDash e_2 : T}{\Gamma_1, \Gamma_2 \vDash e_1; e_2 : T} \qquad \Gamma \vDash b : \mathbb{B}_{\text{sem}}$$

$$\frac{\Gamma_1 \vDash e : \mathbb{B}_{\text{sem}} \qquad \Gamma_2 \vDash e_1 : T \qquad \Gamma_2 \vDash e_2 : T}{\Gamma_1, \Gamma_2 \vDash \text{if } e \text{ then } e_1 \text{ else } e_2 : T} \qquad \emptyset \vDash n : \mathbb{N}_{\text{sem}} \qquad \frac{\Gamma_1 \vDash e_1 : \mathbb{N}_{\text{sem}} \qquad \Gamma_2 \vDash e_2 : \mathbb{N}_{\text{sem}}}{\Gamma_1, \Gamma_2 \vDash e_1 + e_2 : \mathbb{N}_{\text{sem}}}$$

$$\frac{\Gamma_1 \vDash e : \mathbb{N}_{\text{sem}} \qquad \Gamma_2 \vDash e_0 : T \qquad x : T \vDash e_S : T}{\Gamma_1, \Gamma_2 \vDash \mathsf{iter}\ e_0\ e\ (\lambda\, x.e_S) \vDash T} \qquad \frac{\Gamma, x : T \vDash e : U}{\Gamma \vDash \lambda\, x.e : T \multimap U}$$

$$\frac{\Gamma_1 \vDash e_1 : T \multimap U \qquad \Gamma_2 \vDash e_2 : T}{\Gamma_1, \Gamma_2 \vDash e_1(e_2) : U} \qquad \frac{\Gamma_1 \vDash e_1 : T \qquad \Gamma_2 \vDash e_2 : U}{\Gamma_1, \Gamma_2 \vDash (e_1, e_2) : T \otimes U}$$

$$\frac{\Gamma_1 \vDash e_1 : T_1 \otimes T_2 \qquad \Gamma_1, x : T_1, y : T_2 \vDash e_2 : U}{\Gamma_1, \Gamma_2 \vDash \text{let } (x, y) := e_1 \text{ in } e_2 : U} \qquad \emptyset \vDash \mathsf{chan}() : \mathsf{Get}(T) \otimes \mathsf{Put}(T)$$

$$\frac{\Gamma_1 \vDash e_1 : \mathsf{Get}(T) \qquad \Gamma_2 \vDash e_2 : T \multimap \mathbb{1}_{\text{sem}}}{\Gamma_1, \Gamma_2 \vDash \mathsf{get}(e_1, e_2) : \mathbb{1}_{\text{sem}}} \qquad \frac{\Gamma_1 \vDash e_1 : \mathsf{Put}(T) \qquad \Gamma_2 \vDash e_2 : T}{\Gamma_1, \Gamma_2 \vDash \mathsf{put}(e_1, e_2) : \mathbb{1}_{\text{sem}}}$$

Figure 6: The compatibility lemmas

Moreover, the interpretations of $\mathsf{Get}(\cdot)$ and $\mathsf{Put}(\cdot)$ include one time-credit which is used (with the rule TCWP-BURN-CREDIT) to remove the later we obtain when opening the invariant.

We can now define the semantic typing relation. The only non-standard thing is that we existentially quantify over the number of time-credits initially available. Here, in the semantic interpretation, we use as contexts Γ_1, Γ_2 finite maps from strings to semantic types SemType.

$$\mathcal{G}[\![\Gamma]\!] \triangleq \lambda\gamma.\ \ast_{x:T\in\Gamma}\ \gamma(x) \text{ is defined} * T(\gamma(x))$$
$$\Gamma \vDash e : T \triangleq \exists\alpha.\ \$\alpha \twoheadrightarrow \forall\gamma.\ \mathcal{G}[\![\Gamma]\!](\gamma) \twoheadrightarrow \mathcal{E}[\![T]\!](\gamma(e))$$

Soundness We use the traditional approach to proving soundness of a logical relation. We show that in each of the rules, we can replace the syntactic notion ($\vdash$) with the semantic notion ($\vDash$). Specifically, we prove the lemmas presented in Figure 6.

We can us the logical relation and Theorem 7 to derive the termination result:

Lemma 8. *Assume that we are working with a step-index type validating the small existential property. If* $\emptyset \vDash e : T$*, then SN* $(\rightarrow_{\mathsf{t}})\ [e]; \sigma$ *for any* σ*.*

12.3 Adding Impredicative Polymorphism

To add polymorphism to the language, we embed the usual introduction and elimination primitives as derived forms of the runtime term language:

$$\Lambda.e \triangleq \lambda\,().e \qquad e\langle\rangle \triangleq e() \qquad \text{unpack } e \text{ as } x \text{ in } e' \triangleq (\lambda\, x.e')e \qquad \text{pack } e \triangleq e$$

$$x : A \vdash x : A \qquad \frac{\Delta; \Gamma \vdash e : B}{\Delta; \Gamma, x : A \vdash e : B} \qquad \Delta; \emptyset \vdash () : \mathbb{1} \qquad \frac{\Delta; \Gamma_1 \vdash e_1 : \mathbb{1} \qquad \Delta; \Gamma_2 \vdash e_2 : A}{\Delta; \Gamma_1, \Gamma_2 \vdash e_1; e_2 : A}$$

$$\Delta; \Gamma \vdash b : \mathbb{B} \qquad \frac{\Delta; \Gamma_1 \vdash e : \mathbb{B} \qquad \Delta; \Gamma_2 \vdash e_1 : A \qquad \Delta; \Gamma_2 \vdash e_2 : A}{\Delta; \Gamma_1, \Gamma_2 \vdash \text{if } e \text{ then } e_1 \text{ else } e_2 : A} \qquad \Delta; \emptyset \vdash n : \mathbb{N}$$

$$\frac{\Delta; \Gamma_1 \vdash e_1 : \mathbb{N} \qquad \Delta; \Gamma_2 \vdash e_2 : \mathbb{N}}{\Delta; \Gamma_1, \Gamma_2 \vdash e_1 + e_2 : \mathbb{N}} \qquad \frac{\Delta; \Gamma_1 \vdash e : \mathbb{N} \qquad \Delta; \Gamma_2 \vdash e_0 : A \qquad \Delta; x : A \vdash e_S : A}{\Delta; \Gamma_1, \Gamma_2 \vdash \text{iter } e_0\ e\ (\lambda\, x.e_S) \vdash A}$$

$$\frac{\Delta; \Gamma, x : A \vdash e : B}{\Delta; \Gamma \vdash \lambda\, x.e : A \multimap B} \qquad \frac{\Delta; \Gamma_1 \vdash e_1 : A \multimap B \qquad \Delta; \Gamma_2 \vdash e_2 : A}{\Delta; \Gamma_1, \Gamma_2 \vdash e_1(e_2) : B}$$

$$\frac{\Delta; \Gamma_1 \vdash e_1 : A \qquad \Delta; \Gamma_2 \vdash e_2 : B}{\Delta; \Gamma_1, \Gamma_2 \vdash (e_1, e_2) : A \otimes B} \qquad \frac{\Delta; \Gamma_1 \vdash e_1 : A_1 \otimes A_2 \qquad \Delta; \Gamma_1, x : A_1, y : A_2 \vdash e_2 : B}{\Delta; \Gamma_1, \Gamma_2 \vdash \text{let } (x, y) := e_1 \text{ in } e_2 : B}$$

$$\Delta; \emptyset \vdash \mathsf{chan}() : \mathsf{Get}(A) \otimes \mathsf{Put}(A) \qquad \frac{\Delta; \Gamma_1 \vdash e_1 : \mathsf{Get}(A) \qquad \Delta; \Gamma_2 \vdash e_2 : A \multimap \mathbb{1}}{\Delta; \Gamma_1, \Gamma_2 \vdash \mathsf{get}(e_1, e_2) : \mathbb{1}}$$

$$\frac{\Delta; \Gamma_1 \vdash e_1 : \mathsf{Put}(A) \qquad \Delta; \Gamma_2 \vdash e_2 : A}{\Delta; \Gamma_1, \Gamma_2 \vdash \mathsf{put}(e_1, e_2) : \mathbb{1}} \qquad \frac{\Delta, \alpha; \Gamma \vdash e : A \qquad \Delta \vdash \Gamma \qquad \alpha \notin \Delta}{\Delta; \Gamma \vdash \Lambda.e : \Pi\alpha.A}$$

$$\frac{\Delta; \Gamma \vdash e : \Pi\alpha.A}{\Delta; \Gamma \vdash e\langle\rangle : A[B/\alpha]} \qquad \frac{\Delta; \Gamma \vdash e : A[B/\alpha]}{\Delta; \Gamma \vdash \text{pack } e : \exists\alpha.A}$$

$$\frac{\Delta; \Gamma_1 \vdash e : \exists\alpha.A \qquad \Delta, \alpha; \Gamma_2, x : A \vdash e_2 : B \qquad \Delta \vdash B \qquad \Delta \vdash \Gamma_2 \qquad \alpha \notin \Delta}{\Delta; \Gamma_1, \Gamma_2 \vdash \text{unpack } e \text{ as } x \text{ in } e_2 : B}$$

Figure 7: Polymorphic linear type-system for λ_{CHAN}.

This brings us to the linear type system with polymorphism. We use types containing type variables α, we use a linear typing context Γ over these types, and now additionally a type variable context Δ. We define the type well-formedness judgement:

$$\frac{T \text{ is closed under } \Delta}{\Delta \vdash T} \qquad \frac{\forall x : T \in \Gamma.\Delta \vdash T}{\Delta \vdash \Gamma}$$

The type system is then given in Figure 7.

Semantic Interpetation With impredicative polymorphism, our types can now depend on variables. We reflect this in the semantic interpretation by additionally parameterizing the types over semantic types. That is, we consider polymorphic semantic types PolySemType, predicates from finite maps of semantic types to semantic types $\mathbf{T}, \mathbf{U} : (\mathsf{id} \xrightarrow{\text{fin}} \mathsf{SemType}) \to \mathsf{SemType}$. We lift the type formers of the monomorphic case to

the polymorphic case:

$$
\begin{aligned}
\mathbb{1}_{\text{sem, poly}} &\triangleq \lambda\_.\, \mathbb{1}_{\text{sem}} \\
\mathbb{B}_{\text{sem, poly}} &\triangleq \lambda\_.\mathbb{B}_{\text{sem}} \\
\mathbb{N}_{\text{sem, poly}} &\triangleq \lambda\_.\mathbb{N}_{\text{sem}} \\
\mathbf{T} \multimap \mathbf{U} &\triangleq \lambda\delta.\mathbf{T}(\delta) \multimap \mathbf{U}(\delta) \\
\mathbf{T} \otimes \mathbf{U} &\triangleq \lambda\delta.\mathbf{T}(\delta) \otimes \mathbf{U}(\delta) \\
\mathsf{Get}(\mathbf{T}) &\triangleq \lambda\delta.\mathsf{Get}(\mathbf{T}(\delta)) \\
\mathsf{Put}(\mathbf{T}) &\triangleq \lambda\delta.\mathsf{Put}(\mathbf{T}(\delta)) \\
\Pi\alpha\mathbf{T} &\triangleq \lambda\delta.\ \lambda f.\forall\mathbf{U}.\ \mathcal{E}[\![(\mathbf{T}[\mathbf{U}/\alpha])(\delta)]\!](f()) \\
\exists\alpha.\mathbf{T} &\triangleq \lambda\delta.\lambda v.\exists\mathbf{U}.\ (\mathbf{T}[\mathbf{U}/\alpha])(\delta)(v) \\
\mathcal{E}[\![\mathbf{T}]\!]_\delta &\triangleq \lambda e.\mathsf{seqtwp}\ e\ \{v.\ \mathbf{T}(\delta(v))\}
\end{aligned}
$$

where we define $\mathbf{T}[\mathbf{U}/\alpha] \triangleq \lambda\delta.\mathbf{T}(\delta[\alpha := \mathbf{U}(\delta)])$

Semantically, we use contexts $\mathbf{\Gamma} : \mathsf{id} \xrightarrow{\text{fin}} \mathsf{PolySemType}$. We define the semantic typing relation as:

$$
\begin{aligned}
\mathcal{D}[\![\Delta]\!] &\triangleq \lambda\delta : \mathsf{id} \xrightarrow{\text{fin}} \mathsf{SemType}.\ \mathop{\Large *}_{\alpha\in\Delta} \delta(\alpha) \text{ is defined} \\
\mathcal{G}[\![\mathbf{\Gamma}]\!]_\delta &\triangleq \lambda\gamma.\ \mathop{\Large *}_{x:\mathbf{T}\in\mathbf{\Gamma}} \gamma(x) \text{ is defined} * \mathbf{T}(\delta)(\gamma(x)) \\
\delta =_\Delta \delta' &\triangleq \forall(x \mapsto \mathbf{T}) \in \Delta.\ \delta(x) = \delta'(x) \\
\Delta \vDash \mathbf{T} &\triangleq \forall\delta,\delta'.\delta =_\Delta \delta' \Rightarrow \mathbf{T}(\delta) = \mathbf{T}(\delta') \\
\Delta \vDash \mathbf{\Gamma} &\triangleq \forall(x \mapsto \mathbf{T}) \in \mathbf{\Gamma}.\Delta \vDash \mathbf{T} \\
\Delta;\mathbf{\Gamma} \vDash e : \mathbf{T} &\triangleq \exists\alpha.\ \$\alpha \twoheadrightarrow \forall\delta,\gamma.\ \mathcal{D}[\![\Delta]\!](\delta) \twoheadrightarrow \mathcal{G}[\![\mathbf{\Gamma}]\!]_\delta(\gamma) \twoheadrightarrow \mathcal{E}[\![\mathbf{T}]\!]_\delta(\gamma(e))
\end{aligned}
$$

Soundness We use the traditional approach to proving soundness of a logical relation. We show that in each of the rules, we can replace the syntactic notion ($\vdash$) with the semantic notion ($\vDash$). Specifically, we prove the lemmas presented in Figure 8. We can use the logical relation and Theorem 7 to derive the termination result:

Lemma 9. *Assume that we are working with a step-index type validating the small existential property. If* $\emptyset;\emptyset \vDash e : \mathbf{T}$*, then SN* $(\rightarrow_{\mathtt{t}})\ [e];\sigma$ *for any* σ*.*

$$x : \mathbf{T} \vDash x : \mathbf{T} \qquad \frac{\Delta; \mathbf{\Gamma} \vDash e : \mathbf{U}}{\Delta; \mathbf{\Gamma}, x : \mathbf{T} \vDash e : \mathbf{U}} \qquad \Delta; \emptyset \vDash () : \mathbb{1} \qquad \frac{\Delta; \mathbf{\Gamma}_1 \vDash e_1 : \mathbb{1} \qquad \Delta; \mathbf{\Gamma}_2 \vDash e_2 : \mathbf{T}}{\Delta; \mathbf{\Gamma}_1, \mathbf{\Gamma}_2 \vDash e_1; e_2 : \mathbf{T}}$$

$$\Delta; \mathbf{\Gamma} \vDash b : \mathbb{B} \qquad \frac{\Delta; \mathbf{\Gamma}_1 \vDash e : \mathbb{B} \qquad \Delta; \mathbf{\Gamma}_2 \vDash e_1 : \mathbf{T} \qquad \Delta; \mathbf{\Gamma}_2 \vDash e_2 : \mathbf{T}}{\Delta; \mathbf{\Gamma}_1, \mathbf{\Gamma}_2 \vDash \mathsf{if}\ e\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 : \mathbf{T}} \qquad \Delta; \emptyset \vDash n : \mathbb{N}$$

$$\frac{\Delta; \mathbf{\Gamma}_1 \vDash e_1 : \mathbb{N} \qquad \Delta; \mathbf{\Gamma}_2 \vDash e_2 : \mathbb{N}}{\Delta; \mathbf{\Gamma}_1, \mathbf{\Gamma}_2 \vDash e_1 + e_2 : \mathbb{N}} \qquad \frac{\Delta; \mathbf{\Gamma}_1 \vDash e : \mathbb{N} \qquad \Delta; \mathbf{\Gamma}_2 \vDash e_0 : \mathbf{T} \qquad \Delta; x : \mathbf{T} \vDash e_S : \mathbf{T}}{\Delta; \mathbf{\Gamma}_1, \mathbf{\Gamma}_2 \vDash \mathsf{iter}\ e_0\ e\ (\lambda\, x.e_S) \vDash \mathbf{T}}$$

$$\frac{\Delta; \mathbf{\Gamma}, x : \mathbf{T} \vDash e : \mathbf{U}}{\Delta; \mathbf{\Gamma} \vDash \lambda\, x.e : \mathbf{T} \multimap \mathbf{U}} \qquad \frac{\Delta; \mathbf{\Gamma}_1 \vDash e_1 : \mathbf{T} \multimap \mathbf{U} \qquad \Delta; \mathbf{\Gamma}_2 \vDash e_2 : \mathbf{T}}{\Delta; \mathbf{\Gamma}_1, \mathbf{\Gamma}_2 \vDash e_1(e_2) : \mathbf{U}}$$

$$\frac{\Delta; \mathbf{\Gamma}_1 \vDash e_1 : \mathbf{T} \qquad \Delta; \mathbf{\Gamma}_2 \vDash e_2 : \mathbf{U}}{\Delta; \mathbf{\Gamma}_1, \mathbf{\Gamma}_2 \vDash (e_1, e_2) : \mathbf{T} \otimes \mathbf{U}} \qquad \frac{\Delta; \mathbf{\Gamma}_1 \vDash e_1 : \mathbf{T}_1 \otimes \mathbf{T}_2 \qquad \Delta; \mathbf{\Gamma}_1, x : \mathbf{T}_1, y : \mathbf{T}_2 \vDash e_2 : \mathbf{U}}{\Delta; \mathbf{\Gamma}_1, \mathbf{\Gamma}_2 \vDash \mathsf{let}\ (x, y) := e_1\ \mathsf{in}\ e_2 : \mathbf{U}}$$

$$\Delta; \emptyset \vDash \mathsf{chan}() : \mathsf{Get}(\mathbf{T}) \otimes \mathsf{Put}(\mathbf{T}) \qquad \frac{\Delta; \mathbf{\Gamma}_1 \vDash e_1 : \mathsf{Get}(\mathbf{T}) \qquad \Delta; \mathbf{\Gamma}_2 \vDash e_2 : \mathbf{T} \multimap \mathbb{1}}{\Delta; \mathbf{\Gamma}_1, \mathbf{\Gamma}_2 \vDash \mathsf{get}(e_1, e_2) : \mathbb{1}}$$

$$\frac{\Delta; \mathbf{\Gamma}_1 \vDash e_1 : \mathsf{Put}(\mathbf{T}) \qquad \Delta; \mathbf{\Gamma}_2 \vDash e_2 : \mathbf{T}}{\Delta; \mathbf{\Gamma}_1, \mathbf{\Gamma}_2 \vDash \mathsf{put}(e_1, e_2) : \mathbb{1}} \qquad \frac{\Delta, \alpha; \mathbf{\Gamma} \vDash e : \mathbf{T} \qquad \Delta \vDash \mathbf{\Gamma} \qquad \alpha \notin \Delta}{\Delta; \mathbf{\Gamma} \vDash \Lambda.e : \Pi\alpha.\mathbf{T}}$$

$$\frac{\Delta; \mathbf{\Gamma} \vDash e : \Pi\alpha.\mathbf{T}}{\Delta; \mathbf{\Gamma} \vDash e\langle\rangle : \mathbf{T}[\mathbf{U}/\alpha]} \qquad \frac{\Delta; \mathbf{\Gamma} \vDash e : \mathbf{T}[\mathbf{U}/\alpha]}{\Delta; \mathbf{\Gamma} \vDash \mathsf{pack}\ e : \exists\alpha.\mathbf{T}}$$

$$\frac{\Delta; \mathbf{\Gamma}_1 \vDash e : \exists\alpha.\mathbf{T} \qquad \Delta, \alpha; \mathbf{\Gamma}_2, x : \mathbf{T} \vDash e_2 : \mathbf{U} \qquad \Delta \vDash \mathbf{U} \qquad \Delta \vDash \mathbf{\Gamma}_2 \qquad \alpha \notin \Delta}{\Delta; \mathbf{\Gamma}_1, \mathbf{\Gamma}_2 \vDash \mathsf{unpack}\ e\ \mathsf{as}\ x\ \mathsf{in}\ e_2 : \mathbf{U}}$$

Figure 8: Polymorphic compatibility lemmas.

References

Peter Aczel. The type theoretic interpretation of constructive set theory. In Angus Macintyre, Leszek Pacholski, and Jeff Paris, editors, *Logic Colloquium '77*, volume 96 of *Studies in Logic and the Foundations of Mathematics*, pages 55 – 66. Elsevier, 1978. doi: https://doi.org/10.1016/S0049-237X(08)71989-X. URL http://www.sciencedirect.com/science/article/pii/S0049237X0871989X.

Pierre America and Jan Rutten. Solving reflexive domain equations in a category of complete metric spaces. *JCSS*, 39(3):343–375, 1989. doi: 10.1016/0022-0000(89)90027-5.

Lars Birkedal, Kristian Støvring, and Jacob Thamsborg. The category-theoretic solution of recursive metric-space equations. *TCS*, 411(47):4102–4122, 2010. doi: 10.1016/j.tcs.2010.07.010. URL http://dx.doi.org/10.1016/j.tcs.2010.07.010.

Lars Birkedal, Rasmus Ejlers Møgelberg, Jan Schwinghammer, and Kristian Støvring. First steps in synthetic guarded domain theory: Step-indexing in the topos of trees. In *LICS*, pages 55–64, 2011. doi: 10.1109/LICS.2011.16.

Peter Dybjer. A general formulation of simultaneous inductive-recursive definitions in type theory. *The Journal of Symbolic Logic*, 65(2):525–549, 2000. ISSN 00224812. URL http://www.jstor.org/stable/2586554.

Peter Hancock, Conor McBride, Neil Ghani, Lorenzo Malatesta, and Thorsten Altenkirch. Small induction recursion. In Masahito Hasegawa, editor, *Typed Lambda Calculi and Applications*, pages 156–172, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. ISBN 978-3-642-38946-7.

Jonas Kastberg Hinrichsen, Jesper Bengtson, and Robbert Krebbers. Actris: Session-type based reasoning in separation logic. *Proc. ACM Program. Lang.*, 4(POPL), December 2019. doi: 10.1145/3371074. URL https://doi.org/10.1145/3371074.

Dominik Kirst. Formalised set theory: Well-orderings and the axiom of choice, 2014. URL https://www.ps.uni-saarland.de/~kirst/bachelor.php.

Dominik Kirst and Gert Smolka. Categoricity results and large model constructions for second-order ZF in dependent type theory. *Journal of Automated Reasoning*, 2018. First Online: 11 October 2018.

Martin H. Löb. Solution of a problem of leon henkin. *The Journal of Symbolic Logic*, 20(2):115–118, 1955. ISSN 00224812. URL http://www.jstor.org/stable/2266895.

Glen Mével, Jacques-Henri Jourdan, and François Pottier. Time credits and time receipts in Iris. In *ESOP*, volume 11423 of *LNCS*, pages 3–29, 2019. doi: 10.1007/978-3-030-17184-1_1.

R.M. Smullyan and M. Fitting. *Set Theory and the Continuum Problem*. Dover books on mathematics. Dover Publications, 2010. ISBN 9780486474847.

Simon Spies, Neel Krishnaswami, and Derek Dreyer. Transfinite step-indexing for termination, 2021. To appear in POPL'21.

Kasper Svendsen, Filip Sieczkowski, and Lars Birkedal. Transfinite step-indexing: Decoupling concrete and logical steps. In *ESOP*, volume 9632 of *LNCS*, pages 727–751, 2016. doi: 10.1007/978-3-662-49498-1_28.

Benjamin Werner. Sets in types, types in sets. In Martín Abadi and Takayasu Ito, editors, *Theoretical Aspects of Computer Software*, pages 530–546, Berlin, Heidelberg, 1997. Springer Berlin Heidelberg. ISBN 978-3-540-69530-1.