A denotationally-based program logic for higher-order store

Frederik Lerbjerg Aagaard[1] Jonathan Sterling[2] Lars Birkedal[3]

Department of Computer Science
Aarhus University
Aarhus, Denmark

Abstract

Separation logic is used to reason locally about stateful programs. State of the art program logics for higher-order store are usually built on top of untyped operational semantics, in part because traditional denotational methods have struggled to simultaneously account for general references and parametric polymorphism. The recent discovery of simple denotational semantics for general references and polymorphism in synthetic guarded domain theory has enabled us to develop Tulip, a higher-order separation logic over the *typed equational theory* of higher-order store for a monadic version of System $\mathbf{F}_{\mu}^{ref}$. The Tulip logic differs from operationally-based program logics in two ways: predicates range over the *meanings* of typed terms in an arbitrary model rather than over the *raw code* of untyped terms, and they are automatically invariant under the equational congruence of higher-order store, which applies even underneath a binder. As a result, "pure" proof steps that conventionally require focusing the Hoare triple on an operational redex are replaced by a simple equational rewrite in Tulip. We have evaluated Tulip against standard examples involving linked lists in the heap, comparing our abstract equational reasoning with more familiar operational-style reasoning. Our main result is the soundness of Tulip, which we establish by constructing a BI-hyperdoctrine over the denotational semantics of $\mathbf{F}_{\mu}^{ref}$ in an impredicative version of synthetic guarded domain theory.

Keywords: denotational semantics, higher-order separation logic, higher-order store, general references, guarded recursion, synthetic guarded domain theory, BI-hyperdoctrine, impredicative polymorphism, recursive types

1 Introduction

State of the art program logics such as Iris [30] and the Verified Software Toolchain [1] typically combine two design decisions: the basic sorts of the logic are generated by the *raw, untyped terms* of the programming language, and constructs of the program logic are defined in terms of operational semantics layered over this raw syntax. In contrast, the original program logic LCF [24,41,39,23] and successors like HOLCF [27] work in a more abstract way: predicates range over *typed denotations* rather than over untyped raw syntax. The difference in abstraction is substantial: whereas a predicate on raw syntax need not even be invariant under α-equivalence, a predicate on denotations is automatically invariant under the *entire* equational theory of the language. Nevertheless, denotationally-based program logics have fallen by the wayside in part because of the historic failure of denotational methods to provide simple enough answers to the questions posed by higher-order effectful programming, *e.g.* the combination of general references, polymorphism, and concurrency; as a result, the benefits of denotations are primarily reaped today in the first-order realm, as in the successful revival of coinductive resumption semantics by Xia *et al.* [47].

[1] Email: aagaard@cs.au.dk
[2] Email: jsterling@cs.au.dk
[3] Email: birkedal@cs.au.dk

Denotational and algebraic approaches to program semantics have other benefits besides their abstractness: they are more modular, and they are more directly compatible with practical mathematical tools from category theory and topology, which have become increasingly relevant in light of recent interest in probabilistic and differentiable computing for the sciences. We therefore have ample motivation to bridge the gap between denotational methods and modern logics for reasoning about programs.

1.1 Denotational semantics of general references and polymorphism

Sterling *et al.* [46] have recently discovered a simple denotational semantics for general references and polymorphism, decomposing the problem of higher-order store with semantic worlds into two separate constructs that are easily combined: impredicativity and guarded recursion. At the heart of their development is ***Impredicative Guarded Dependent Type Theory*** (**iGDTT**), a type theoretic metalanguage for synthetic guarded domain theory [11] extended by an impredicative universe of sets. Although *op. cit.* have made promising strides in the denotational understanding of higher-order stateful programs, non-trivial *reasoning* about state requires more than equational logic. Operationally-based program logics like Iris employ ***guarded higher-order separation logic*** for this purpose; our contribution in this paper is to adapt these methods to the denotational semantics of Sterling *et al.*

1.2 Higher-order separation logic over denotational semantics

We extend the *monadic equational theory* of higher-order store for System $\mathbf{F}_\mu^{ref}$ to a full higher-order separation logic called Tulip. In contrast to systems like Iris which layer the separation logic over an encoding of the *raw syntax* of the object language, predicates in Tulip range not over code but rather over the actual types of $\mathbf{F}_\mu^{ref}$, subject to the full equational congruence. In a conventional operationally-based program logic, equational reasoning is rather only applicable within the focus of a Hoare triple and is moreover limited to operational head reduction and cannot occur under binders, *etc.*

Our main contribution is the soundness of Tulip for reasoning about Sterling *et al.*'s denotational semantics of $\mathbf{F}_\mu^{ref}$, which we establish by constructing a suitable BI-hyperdoctrine [8]. We have made pervasive use of type theoretic internal languages to facilitate this work: first, System $\mathbf{F}_\mu^{ref}$ is intepreted in *internal co-presheaves* on a poset of semantic heap layouts $\mathbb{W}$ defined in **iGDTT** by solving a guarded recursive domain equation; then, our BI-hyperdoctrine is itself constructed in the type theoretic language of co-presheaves on $\mathbb{W}$. Because **iGDTT** itself is sound [46], it follows that our logic is sound.

1.3 Structure of this paper

In Section 2 we describe the Tulip logic and its computational substrate $\mathbf{F}_\mu^{ref}$; in Section 3, we illustrate the use of Tulip through a worked example involving linked lists in the heap. In Section 4.1, we recall the **iGDTT** metalanguage and the basics of synthetic guarded domain theory, which we use in Sections 4.2 and 4.3 to explain the denotational semantics of higher-order store and the interpretation of guarded higher-order separation logic over this model. In Section 4.4, we show that Tulip is sound with respect to this model and therefore consistent.

2 Tulip: an equational program logic for higher-order store

In this section, we introduce a Tulip program logic that extends the monadic equational logic of System $\mathbf{F}_\mu^{ref}$ with the connectives and rules of guarded higher-order separation logic [13], together with a built-in connective for *weakest preconditions*. The Tulip logic is related to System $\mathbf{F}_\mu^{ref}$ in roughly the same way that **LCF** relates to **PCF** [41,44].

Tulip will have two kinds of types — ***"program types"*** and ***"logical types"*** — and to efficiently organize the formalism, we will define the typehood judgment schematically in a flag $\iota \in \{\boldsymbol{p}, \boldsymbol{\ell}\}$.[4] With

[4] Both System F *and* higher-order logic are modeled by impredicative universes, but it is well-known that one impredicative universe cannot contain another [17], which leads us to introduce the stratification of logical types from program types. The prime example of a logical type that is not a program type will be the type of all propositions.

this convention in hand, the forms of judgment of Tulip classify type contexts $\Xi\ \mathit{tctx}$, element contexts $\Xi \vdash \Gamma\ \mathit{ctx}$, types $\Xi \vdash A\ \mathit{type} @ \iota$, program elements $\Xi \mid \Gamma \vdash a : A$, propositions $\Xi \mid \Gamma \vdash \phi\ \mathit{prop}$, and entailments $\Xi \mid \Gamma \mid \phi \vdash \psi$. Every syntactic construct of Tulip is subject to a relation of ***judgmental equality*** ($\equiv$) which is respected by all judgments and preserved by all operations; for instance, we shall write $\Xi \mid \Gamma \vdash u \equiv v : A$ to mean that u and v are judgmentally equal elements of type A.

Notation 2.1 *When specifying the rules of* Tulip*, we will often write entailments* $\Xi \mid \Gamma \mid \phi \vdash \psi$ *as* $\phi \vdash \psi$ *when the contexts do not change from premise to conclusion.*

2.1 Rules for types and elements

Type contexts $\Xi\ \mathit{tctx}$ contain variables α ranging over *program types* only. Element contexts $\Xi \vdash \Gamma\ \mathit{ctx}$ contain variables $x : A$ ranging over elements of types $\Xi \vdash A\ \mathit{type} @ \iota$. Both program types and logical types are closed under function spaces, cartesian products, inductive types, and universal quantification over program types; program types are furthermore closed under a computational monad in the sense of Moggi [36], as well as existential and recursive types. The only subtlety is that the `unfold` destructor for recursive types is treated *effectfully*, confining recursion to the monad. Every program type is a logical type, and the inclusion of program types into logical types commutes up to isomorphism with function spaces, product types, and inductive types. We will treat these coercions silently as it causes no ambiguity.

For lack of space, we will not dwell on the product and function types except to comment that they satisfy the full universal properties of exponentials and cartesian products up to judgmental equality. Likewise, we do not present here the (quite standard) account of inductive types — except to note that we close both program and logical types under inductives, and structurally recursive functions into logical types on program-level data are permitted.

2.1.1 The computational monad

The universe of program types is closed under a computational monad $\mathtt{T}$ in the sense of Moggi [36] governing both state and general recursion.

$$\frac{\Xi \vdash A\ \mathit{type} @ \boldsymbol{p}}{\Xi \vdash \mathtt{T}\,A\ \mathit{type} @ \boldsymbol{p}} \qquad \frac{\Xi \mid \Gamma \vdash u : A}{\Xi \mid \Gamma \vdash \mathtt{ret}\,u : \mathtt{T}\,A} \qquad \frac{\Xi \mid \Gamma \vdash u : \mathtt{T}\,A \qquad \Xi \mid \Gamma, x : A \vdash v : \mathtt{T}\,B}{\Xi \mid \Gamma \vdash x \leftarrow u; v : \mathtt{T}\,B}$$

We present the standard equational theory of monads in Appendix A. We defer our discussion of the monadic operations for general recursion and state to Section 2.1.2.

2.1.2 General recursion and general references

Program types are closed under both general recursive types and general reference types. As the rules for these are somewhat subtle in our monadic environment, we cover them in detail. First we describe the formation rules for types and terms, before introducing their equational theory.

$$\frac{\Xi, \alpha \vdash A\ \mathit{type} @ \boldsymbol{p}}{\Xi \vdash \mu\alpha.A\ \mathit{type} @ \boldsymbol{p}} \qquad \frac{\Xi \mid \Gamma \vdash u : A[\mu\alpha.A/\alpha]}{\Xi \mid \Gamma \vdash \mathtt{fold}\,u : \mu\alpha.A} \qquad \frac{\Xi \mid \Gamma \vdash u : \mu\alpha.A}{\Xi \mid \Gamma \vdash \mathtt{unfold}\,u : \mathtt{T}\,A[\mu\alpha.A/\alpha]}$$

$$\frac{\Xi \vdash A\ \mathit{type} @ \boldsymbol{p}}{\Xi \vdash \mathtt{ref}\,A\ \mathit{type} @ \boldsymbol{p}} \qquad \frac{\Xi \mid \Gamma \vdash u : \mathtt{ref}\,A}{\Xi \mid \Gamma \vdash \mathtt{get}\,u : \mathtt{T}\,A} \qquad \frac{\Xi \mid \Gamma \vdash u : \mathtt{ref}\,A \qquad \Xi \mid \Gamma \vdash v : A}{\Xi \mid \Gamma \vdash \mathtt{set}\,u\,v : \mathtt{T}\,()}$$

$$\frac{\Xi \mid \Gamma \vdash u : A}{\Xi \mid \Gamma \vdash \mathtt{new}\,u : \mathtt{T}\,(\mathtt{ref}\,A)} \qquad \frac{}{\Xi \mid \Gamma \vdash \mathtt{step} : \mathtt{T}\,()}$$

The `step` constructor above is a "no-op" instruction witnessing the *guarded* or *intensional* nature of our denotational semantics;[5] this is reflected in the equational theory by the emission of `step` instructions

[5] In the presence of polymorphism, the guarded interpretation of state and general recursion seems to be forced [12].

with reads to the heap and the unfolding of recursive types. Our logic, as we explain later, will provide rules that allow these steps to be used as fuel for recursive deductions.

UNFOLD-OF-FOLD
$$\vdash \mathtt{unfold}\,(\mathtt{fold}\,u) \equiv \mathtt{step};\mathtt{ret}\,u : \mathtt{T}\,A[\mu\alpha.A/\alpha]$$

FOLD-OF-UNFOLD
$$\vdash x \leftarrow \mathtt{unfold}\,u;\mathtt{ret}\,(\mathtt{fold}\,x) \equiv \mathtt{step};\mathtt{ret}\,u : \mathtt{T}\,(\mu\alpha.A)$$

GET-AFTER-SET
$$\vdash \mathtt{set}\,u\,v;\mathtt{get}\,u \equiv \mathtt{step};\mathtt{set}\,u\,v;\mathtt{ret}\,v : \mathtt{T}\,A$$

SET-AFTER-NEW
$$\vdash (x \leftarrow \mathtt{new}\,u;\mathtt{set}\,x\,v;w) \equiv (x \leftarrow \mathtt{new}\,v;w) : \mathtt{T}\,B$$

SET-AFTER-SET
$$\vdash \mathtt{set}\,u\,v;\mathtt{set}\,u\,w \equiv \mathtt{set}\,u\,w : \mathtt{T}\,A$$

GET-AFTER-GET
$$\vdash (x \leftarrow \mathtt{get}\,u; y \leftarrow \mathtt{get}\,v; w) \equiv (y \leftarrow \mathtt{get}\,v; x \leftarrow \mathtt{get}\,u; w) : \mathtt{T}\,C$$

SET-AFTER-GET
$$\vdash (x \leftarrow \mathtt{get}\,u;\mathtt{set}\,u\,x;w) \equiv (x \leftarrow \mathtt{get}\,u;w) : \mathtt{T}\,B$$

We also have, but do not present, rules that commute `step` past all primitive effects.

Remark 2.2 Observe that ours is a theory of *higher-order global store* rather than *higher-order local store* in the sense that allocations are not hidden but rather have a globally observable effect. The state of the art in denotational semantics for local store is currently restricted to references of *ground type* [31].

2.1.3 Universal and existential types

Our language contains both universal and existential types; it is common to "encode" the latter in terms of the former, but encodings of this form are not quite correct as they neglect the equational theory of existential types. [6] We therefore include both connectives as primitives.

$$\frac{\Xi, \alpha \vdash A\ \mathit{type} \mathbin{@} \iota}{\Xi \vdash \forall\alpha.A\ \mathit{type} \mathbin{@} \iota} \qquad \frac{\Xi, \alpha \vdash A\ \mathit{type} \mathbin{@} \boldsymbol{p}}{\Xi \vdash \exists\alpha.A\ \mathit{type} \mathbin{@} \boldsymbol{p}} \qquad \frac{\Xi, \alpha \mid \Gamma \vdash u : A}{\Xi \mid \Gamma \vdash \Lambda\alpha.u : \forall\alpha.A} \qquad \frac{\Xi \mid \Gamma \vdash u : \forall\alpha.A \qquad \Xi \vdash B\ \mathit{type} \mathbin{@} \boldsymbol{p}}{\Xi \mid \Gamma \vdash u \cdot B : A[B/\alpha]}$$

$$\frac{\Xi \vdash B\ \mathit{type} \mathbin{@} \boldsymbol{p} \qquad \Xi \mid \Gamma \vdash u : A[B/\alpha]}{\Xi \mid \Gamma \vdash \mathtt{pack}\,(B,u) : \exists\alpha.A} \qquad \frac{\Xi \vdash C\ \mathit{type} \mathbin{@} \boldsymbol{p} \qquad \Xi \mid \Gamma \vdash u : \exists\alpha.A \qquad \Xi, \alpha \mid \Gamma, x : A \vdash v : C}{\Xi \mid \Gamma \vdash \mathtt{let}\ \mathtt{pack}\,(\alpha,x) = u\ \mathtt{in}\ v : C}$$

The equational theory of universals and existentials is presented in Appendix A.

2.2 Rules for the propositional fragment

The logical layer of Tulip is a form of *guarded higher-order separation logic*. We will treat each of these aspects modularly; in Sections 2.2.1 and 2.2.2 we recall the rules of intuitionistic higher-order equational logic, and in Section 2.2.3 we recall (affine) separation logic, and we finish in Section 2.2.4 with an overview of the *later modality* and its Löb induction principle, and how they interact with the rest of the logic.

2.2.1 Equational logic

So far the only form of equality that we have considered is *judgmental* or *external equality* in the sense of Jacobs [29, §3.2]; in order to facilitate equational reasoning in the logic, we add propositional equality and relate it to judgmental equality using the following rules:

EQUALITY FORMATION
$$\frac{\Xi \vdash A\ \mathit{type} \mathbin{@} \iota \qquad \Xi \mid \Gamma \vdash u, v : A}{\Xi \mid \Gamma \vdash u =_A v\ \mathit{prop}}$$

Lawvere rule
$$\frac{\Xi \mid \Gamma, x : A \mid \phi[x/y] \vdash \psi[x/y]}{\Xi \mid \Gamma, x : A, y : A \mid \phi \wedge x =_A y \vdash \psi}$$

EQUALITY REFLECTION
$$\frac{\Xi \mid \Gamma \mid \top \vdash u =_A v}{\Xi \mid \Gamma \vdash u \equiv v : A}$$

As Jacobs [29] explains, the Lawvere rule entails all important properties of equality, including congruence for all constructs of Tulip and its $\mathbf{F}_\mu^{ref}$ substrate. The equality reflection rule above is

[6] The correct equational theory of polymorphically-encoded existentials does hold up to parametricity, but parametricity is an emergent property of *syntax*. The purpose of specifying an equational theory is to constrain *all* models, not only the nonstandard parametric models.

needed to complete the relationship between (unconditional) judgmental equality and propositional equality without assumptions. [7]

2.2.2 *Intuitionistic higher-order logic*

We assume the usual rules of intuitionistic first-order logic over logical types; in particular, in addition to implications $\Rightarrow$, conjunctions $\wedge$, and disjunctions $\vee$, we may form universal and existential quantifications $\forall(x : A).\phi$ and $\exists(x : A).\phi$ when A is a logical type. The logic is made higher-order by introducing a logical type classifying all propositions.

$$\frac{}{\Xi \vdash \mathtt{prop}\ \mathit{type}\ @\ \boldsymbol{\ell}} \qquad \frac{\Xi \mid \Gamma \vdash \phi : \mathtt{prop}}{\Xi \mid \Gamma \vdash \phi\ \mathit{prop}} \qquad \frac{\Xi \mid \Gamma \vdash \phi \equiv \psi : \mathtt{prop}}{\Xi \mid \Gamma \vdash \phi \equiv \psi\ \mathit{prop}} \qquad \frac{\Xi \mid \Gamma \mid \chi \wedge \phi \vdash \psi \qquad \Xi \mid \Gamma \mid \chi \wedge \psi \vdash \phi}{\Xi \mid \Gamma \mid \chi \vdash \phi =_{\mathtt{prop}} \psi}$$

2.2.3 *Separation logic for local reasoning*

We assume the standard rules for intuitionistic affine separation logic, in which we have a separating conjunction $\phi * \psi$ with unit $\top$, and separating implications given as right adjoints $(- * \psi) \dashv (\psi \mathrel{-\!\!*} -)$.

$$\frac{\chi * \phi \vdash \psi}{\chi \vdash \phi \mathrel{-\!\!*} \psi} \qquad \frac{\phi \vdash \phi' \qquad \psi \vdash \psi'}{\phi * \psi \vdash \phi' * \psi'} \qquad \begin{array}{l} \phi * \psi \vdash \phi \wedge \psi \\ (\chi * \phi) * \psi \dashv\vdash \chi * (\phi * \psi) \\ \phi * \psi \dashv\vdash \psi * \phi \end{array}$$

In addition to the separating conjunction and implication, separation logic contains a *coreflective sublogic* of ***persistent propositions***, which are to a first approximation those that are not sensitive to the state of the heap and can therefore be duplicated freely. In particular, we add an idempotent comonadic modality $\Box$ that takes a proposition to its "persistent core"; a proposition ϕ is then called persistent if the entailment $\phi \vdash \Box\phi$ holds. Persistent propositions in this sense are closed under all the connectives of intuitionistic first-order logic; we omit the rules that establish this and instead focus on the interaction between persistence and the connectives of separation logic:

$$\frac{\Xi \mid \Gamma \vdash \phi\ \mathit{prop}}{\Xi \mid \Gamma \vdash \Box\phi\ \mathit{prop}} \qquad \frac{\phi \vdash \psi}{\Box\phi \vdash \Box\psi} \qquad \begin{array}{l} \Box\phi \vdash \phi * \Box\phi \\ \Box\phi \vdash \Box\Box\phi \\ \phi \wedge \Box\psi \vdash \phi * \Box\psi \end{array}$$

It follows from the above that the separating conjunction of persistent propositions is their conjunction.

2.2.4 *The later modality and guarded recursion*

In order to use Tulip as a logic to reason about general recursion (including recursion inherent in the heap), it is necessary to introduce the ***later modality*** $\triangleright$; as in prior works [2,21,30], the later modality abstracts away the onerous step-indices of more concrete accounts of higher-order store leaving only the essential logical structure of guarded-recursive reasoning. The abstract will meet the concrete, however, when we illustrate below the interaction between the later modality and the constructions of our programming language in the fold equality and step equality rules.

$$\frac{\Xi \mid \Gamma \vdash \phi\ \mathit{prop}}{\Xi \mid \Gamma \vdash \triangleright\phi\ \mathit{prop}} \qquad \begin{array}{l} \phi \vdash \triangleright\phi \\ \triangleright\phi \wedge \triangleright\psi \vdash \triangleright(\phi \wedge \psi) \end{array} \qquad \begin{array}{l} \triangleright\phi * \triangleright\psi \vdash \triangleright(\phi * \psi) \\ \phi \mathrel{-\!\!*} \psi \vdash \triangleright\phi \mathrel{-\!\!*} \triangleright\psi \\ \triangleright\Box\phi \dashv\vdash \Box\triangleright\phi \end{array}$$

fold equality
$$\frac{\phi \vdash \triangleright(u =_{A[\mu\alpha.A/\alpha]} v)}{\phi \vdash \mathtt{fold}\, u =_{\mu\alpha.A} \mathtt{fold}\, v}$$

step equality
$$\frac{\phi \vdash \triangleright(u =_{\mathtt{T}\,A} v)}{\phi \vdash \mathtt{step}; u =_{\mathtt{T}\,A} \mathtt{step}; v}$$

Löb induction
$$\triangleright\phi \Rightarrow \phi \vdash \phi$$

[7] Ordinarily, equality reflection would imply the Lawvere rule but for the lack of propositional assumptions in judgmental equality.

The Löb induction rule above is what makes (guarded) recursive reasoning possible in Tulip; the function of the fold equality and step equality rules is to provide "fuel" that can be used to discharge the later modality in the Löb induction hypothesis. This is the sense in which Tulip evinces an abstract form of step-indexing: operations that semantically involve unfolding a recursive domain equation leave behind abstract steps that can be used to advance in time in relation to the later modality.

2.2.5 Weakest preconditions

For reasoning about programs, we introduce a connective called the ***partial weakest precondition***. Morally, the weakest precondition of a program and a predicate, as the name suggests, is the weakest proposition that guarantees the predicate shall hold of any return value of the program. Note, however, that, despite the name, we make no claim, neither in the logic nor in its semantics, that it is in fact the weakest such proposition; this is in line with the usage in Iris [30].

WP formation
$$\frac{\Xi \vdash A \ \mathit{type} \ @ \ \boldsymbol{p} \qquad \Xi \mid \Gamma \vdash e : \mathtt{T}\, A \qquad \Xi \mid \Gamma, x : A \vdash \phi \ \mathit{prop}}{\Xi \mid \Gamma \vdash \mathtt{wp}\, e\, \{x.\, \phi\} \ \mathit{prop}}$$

WP-wand
$$(\forall x.\, \phi \mathrel{-\!\!*} \psi) * \mathtt{wp}\, e\, \{x.\, \phi\} \vdash \mathtt{wp}\, e\, \{x.\, \psi\}$$

WP-val
$$\phi[e/x] \vdash \mathtt{wp}\, (\mathtt{ret}\, e)\, \{x.\, \phi\}$$

WP-bind
$$\mathtt{wp}\, e_1\, \{x.\, \mathtt{wp}\, e_2\, \{y.\, \phi\}\} \vdash \mathtt{wp}\, (x \leftarrow e_1; e_2)\, \{y.\, \phi\}$$

WP-get
$$\exists x.\, [\ell \hookrightarrow x] * \triangleright([\ell \hookrightarrow x] \mathrel{-\!\!*} \phi) \vdash \mathtt{wp}\, (\mathtt{get}\, \ell)\, \{x.\, \phi\}$$

WP-set
$$(\exists y.\, [\ell \hookrightarrow y]) * ([\ell \hookrightarrow e] \mathrel{-\!\!*} \phi) \vdash \mathtt{wp}\, (\mathtt{set}\, \ell\, e)\, \{\_.\, \phi\}$$

WP-new
$$\forall x.\, [x \hookrightarrow e] \mathrel{-\!\!*} \phi \vdash \mathtt{wp}\, (\mathtt{new}\, e)\, \{x.\, \phi\}$$

WP-step
$$\triangleright\phi \vdash \mathtt{wp}\, \mathtt{step}\, \{\_.\, \phi\}$$

It is worth comparing the above rules to the rules in Iris [30, Fig.13]. Our first four entailments are exactly the same as the corresponding rules in Iris; note that as in Iris, WP-wand implies the frame rule. Our rules for WP-set and WP-new rules differ slightly from those of Iris, which have an occurrence of the later modality in the antecedent that ours lack. This is because every operation in Iris takes a step, but in our semantics, only operations that semantically correspond to unfolding a recursive domain equation do.

Values in the heap are stored "one step in the future", so `get` must take a step before returning; consequently, the postcondition of `get` only needs to hold *later*. However, it is not an issue for `set` and `new` to send a value to the future without going there, implying the postcondition must be known now. This behaviour is reflected in the rules get-after-set and set-after-new, where the former shows that `get` takes a step, whilst `ret` does not, and the latter shows that `set` does not take a step. A second difference to point out is that whilst Iris has a rule for β-reduction of functions, we do not. This is because our programming language is subject to the β/η-equational theory of monadic λ-calculus, so rather than having a rule stating that $\mathtt{wp}\, (e[v/x])\, \{y.\, \phi\} \vdash \mathtt{wp}\, ((\lambda x.\, e)\, v)\, \{y.\, \phi\}$, the two propositions are actually *convertible*.

Example 2.3 It is not difficult to encode the more familiar ***Hoare triples*** in Tulip, using the standard decomposition into persistence, separating implication, and weakest precondition:

$$\{\phi\}\, e\, \{x.\, \psi\} \triangleq \square(\phi \mathrel{-\!\!*} \mathtt{wp}\, e\, \{x.\, \psi\}).$$

2.3 Recursive Functions

As is standard, recursive types can be used to derive recursive terms via self-referential types (see *e.g.* Harper [25]). Using this approach, we obtain terms $\mathtt{rec}\, f(x)\ \mathtt{in}\ e$ for recursive functions typed as follows:

$$\frac{\Xi \vdash A, B \ \mathit{type} \ @ \ \boldsymbol{p} \qquad \Xi \mid \Gamma, f : A \to \mathtt{T}\, B, x : A \vdash e : \mathtt{T}\, B}{\Xi \mid \Gamma \vdash \mathtt{rec}\, f(x)\ \mathtt{in}\ e : A \to \mathtt{T}\, B}$$

This derived form satisfies the equation $(\mathtt{rec}\, f(x)\ \mathtt{in}\ e)\, x \equiv \mathtt{step}; e[\mathtt{rec}\, f(x)\ \mathtt{in}\ e/f]$. Löb induction

implies the following weakest precondition rule, reminiscent of the corresponding Hoare triple rule in Iris [9]:

$$\frac{\begin{array}{l}\Xi \vdash A, B \ \mathit{type} @ \boldsymbol{p} \qquad \Xi \vdash C \ \mathit{type} @ \boldsymbol{\ell} \qquad \Xi \mid \Gamma, z : C, f : A \to \mathtt{T}\, B, x : A \vdash e : \mathtt{T}\, B \\ \Xi \mid \Gamma \vdash \phi \ \mathit{prop} \qquad \Xi \mid \Gamma, z : C, x : A \vdash \psi \ \mathit{prop} \qquad \Xi \mid \Gamma, z : C, x : A, y : B \vdash \chi \ \mathit{prop} \\ \Xi \mid \Gamma \mid \phi \wedge \forall z.\, \forall x.\, \psi \mathrel{-\!\!*} \mathtt{wp}\,((\mathtt{rec}\, f(x)\ \mathtt{in}\ e)\, x)\,\{y.\, \chi\} \vdash \forall z.\, \forall x.\, \psi \mathrel{-\!\!*} \mathtt{wp}\,(e[\mathtt{rec}\, f(x)\ \mathtt{in}\ e/f])\,\{y.\, \chi\}\end{array}}{\Xi \mid \Gamma \mid \triangleright\phi \vdash \forall z.\, \forall x.\, \psi \mathrel{-\!\!*} \mathtt{wp}\,((\mathtt{rec}\, f(x)\ \mathtt{in}\ e)\, x)\,\{y.\, \chi\}}\ \text{WP-REC}$$

3 Case study: verifying the *append* function on linked lists

In this section, we illustrate the use of Tulip by an elementary case study: linked lists in the heap and their append function. The proof is very similar to the one in Iris [9, §4.2], although some steps are perhaps slightly simpler as they use equational reasoning rather than explicit rules for reduction in weakest preconditions. We first define a recursive type of imperative linked lists on any type α:

$$\mathtt{ilist}\,\alpha \triangleq \mu\rho.\mathtt{1} + \mathtt{ref}\,(\alpha \times \rho).$$

Our goal is to define the *append* function on imperative linked lists and prove that it is correct. This means, in particular, to show that it behaves the same as a pure *reference implementation* defined on functional lists. In order to say what it means for a function on imperative lists to behave like a function on linked lists, we must first introduce a formal correspondence between the two types [9, Sec.4.2]. This can be done directly as a structurally recursive function in Tulip.

Construction 3.1 (The list invariant) *We define a correspondence* $(\approx) : \mathtt{ilist}\,\alpha \to \mathtt{list}\,\alpha \to \mathtt{prop}$ *in* Tulip *by structural recursion on the second argument.*

$$\begin{array}{l}(\approx) : \mathtt{ilist}\,\alpha \to \mathtt{list}\,\alpha \to \mathtt{prop} \\ l \approx [\,] \triangleq (l = \mathtt{fold}\,(\mathtt{inl}\,())) \\ l \approx (x :: xs) \triangleq \\ \quad \exists (r : \mathtt{ref}\,(A \times \mathtt{ilist}\,\alpha)). \\ \quad \exists (l' : \mathtt{ilist}\,\alpha). \\ \quad (l = \mathtt{fold}\,(\mathtt{inr}\, r)) * [r \hookrightarrow (x, l')] * (l' \approx xs)\end{array}$$

Note that whilst elements of type $\mathtt{ilist}\,\alpha$ could potentially be cyclic, this is ruled out by the list invariant above: the separating conjunction consumes the location r so it cannot appear again, and furthermore, a cyclic list would be infinite and therefore cannot correspond to a functional list.

Construction 3.2 (The *append* function) *We define the append function on linked lists and its pure reference implementation on functional lists below.*

$$\begin{array}{l}\mathtt{iap} : \mathtt{ilist}\,\alpha \times \mathtt{ilist}\,\alpha \to \mathtt{T}\,(\mathtt{ilist}\,\alpha) \\ \mathtt{iap} \triangleq \\ \quad \mathtt{rec}\, f(l_1, l_2)\ \mathtt{in} \\ \quad\quad z \leftarrow \mathtt{unfold}\, l_1; \\ \quad\quad \mathtt{match}\, z\, \mathtt{with} \\ \quad\quad\quad \mathtt{inl}\ \_ \Rightarrow \mathtt{ret}\, l_2 \\ \quad\quad\quad \mathtt{inr}\, r \Rightarrow (a, l_1') \leftarrow \mathtt{get}\, r; l_3 \leftarrow f\,(l_1', l_2); \mathtt{set}\, r\,(a, l_3); \mathtt{ret}\,(\mathtt{fold}\,(\mathtt{inr}\, r))\end{array}$$

The function above is both impure and general recursive. By contrast, the reference implementation below is pure and structurally recursive.

$$\begin{array}{l}(\oplus) : \mathtt{list}\,\alpha \to \mathtt{list}\,\alpha \to \mathtt{list}\,\alpha \\ [\,] \oplus ys \triangleq ys \\ (x :: xs) \oplus ys \triangleq x :: (xs \oplus ys)\end{array}$$

We can now prove that `iap` behaves according to the reference implementation $(\oplus)$.

Theorem 3.3 *The following sequent is derivable in* Tulip*:*

$$\alpha \mid \cdot \mid \top \vdash \forall (u_1, u_2 : \mathtt{list}\,\alpha; l_1, l_2 : \mathtt{ilist}\,\alpha).\, l_1 \approx u_1 * l_2 \approx u_2 \mathrel{-\!\!*} \mathtt{wp}\,(\mathtt{iap}\,(l_1, l_2))\,\{x.\, x \approx u_1 \oplus u_2\}$$

Proof. Let $[\mathtt{iap}]$ be the function defined so that $\mathtt{iap}\,(l_1, l_2) \equiv \mathtt{rec}\, f(l_1, l_2)\ \mathtt{in}\ [\mathtt{iap}]\, f\,(l_1, l_2)$, and let Q be the following predicate:

$$Q : (\mathtt{ilist}\,\alpha \times \mathtt{ilist}\,\alpha \to \mathtt{T}\,(\mathtt{ilist}\,\alpha)) \to \mathtt{prop}$$
$$Q\,f \triangleq \forall (u_1, u_2 : \mathtt{list}\,\alpha; l_1, l_2 : \mathtt{ilist}\,\alpha).\, l_1 \approx u_1 * l_2 \approx u_2 \mathrel{-\!\!*} \mathtt{wp}\,(f\,(l_1, l_2))\,\{x.\, x \approx u_1 \oplus u_2\}$$

Our goal is to prove $Q\,\mathtt{iap}$; applying the WP-REC rule, it suffices to show that $Q\,\mathtt{iap} \vdash Q\,([\mathtt{iap}]\,\mathtt{iap})$. We proceed by cases on u_1; the only non-trivial case is the following: $Q\,\mathtt{iap} * l_1 \approx v :: vs * l_2 \approx u_2 \vdash \mathtt{wp}\,([\mathtt{iap}]\,\mathtt{iap}\,(l_1, l_2))\,\{x.\, x \approx v :: (vs \oplus u_2)\}$. Rewriting by the defining clause of $(\approx)$, we may assume $r : \mathtt{ref}\,(A \times \mathtt{ilist}\,\alpha)$ and $s : \mathtt{ilist}\,\alpha$ to prove the following:

$$Q\,\mathtt{iap} * (l_2 \approx u_2) * [r \hookrightarrow (v, s)] * (s \approx vs) \vdash \mathtt{wp}\,([\mathtt{iap}]\,\mathtt{iap}\,(\mathtt{fold}\,(\mathtt{inr}\,r), l_2))\,\{x.\, x \approx v :: (vs \oplus u_2)\}$$

Applying equational reasoning (including UNFOLD-OF-FOLD), we convert our goal to the following:

$$Q\,\mathtt{iap} * (l_2 \approx u_2) * [r \hookrightarrow (v, s)] * (s \approx vs) \vdash \mathtt{wp}\,(\mathtt{step}; (a, l_1') \leftarrow \mathtt{get}\,r; l_3 \leftarrow \mathtt{iap}\,(l_1', l_2); \mathtt{set}\,r\,(a, l_3); \mathtt{ret}\,(\mathtt{fold}\,(\mathtt{inr}\,r)))\,\{x.\, x \approx v :: (vs \oplus u_2)\}$$

Applying WP-STEP and the introduction rule for the later modality, it suffices to prove:

$$Q\,\mathtt{iap} * (l_2 \approx u_2) * [r \hookrightarrow (v, s)] * (s \approx vs) \vdash \mathtt{wp}\,((a, l_1') \leftarrow \mathtt{get}\,r; l_3 \leftarrow \mathtt{iap}\,(l_1', l_2); \mathtt{set}\,r\,(a, l_3); \mathtt{ret}\,(\mathtt{fold}\,(\mathtt{inr}\,r)))\,\{x.\, x \approx v :: (vs \oplus u_2)\}$$

Repeatedly applying weakest precondition rules and other administrative rules, it suffices to prove:

$$Q\,\mathtt{iap} * (l_2 \approx u_2) * (s \approx vs) * [r \hookrightarrow (v, s)] \vdash \mathtt{wp}\,(\mathtt{iap}\,(s, l_2))\,\{l_3.\, \mathtt{wp}\,(\mathtt{set}\,r\,(v, l_3))\,\{\_.\, \mathtt{fold}\,(\mathtt{inr}\,r) \approx v :: (vs \oplus u_2)\}\}$$

Next we use $Q\,\mathtt{iap}$ and WP-WAND to reduce our goal as follows, fixing $l_3 : \mathtt{ilist}\,\alpha$:

$$[r \hookrightarrow (v, s)] * (l_3 \approx vs \oplus u_2) \vdash \mathtt{wp}\,(\mathtt{set}\,r\,(v, l_3))\,\{\_.\, \mathtt{fold}\,(\mathtt{inr}\,r) \approx v :: (vs \oplus u_2)\}$$

Applying WP-SET, we arrive at the following goal:

$$[r \hookrightarrow (v, l_3)] * (l_3 \approx vs \oplus u_2) \vdash \mathtt{fold}\,(\mathtt{inr}\,r) \approx v :: (vs \oplus u_2)$$

The above is immediate by definition of $(\approx)$ and instantiation of existential variables. □

4 Denotational semantics of Tulip in impredicative guarded dependent type theory

The denotational semantics of Tulip is an extension of the model of System $\mathbf{F}_\mu^{ref}$ previously constructed by Sterling, Gratzer, and Birkedal [46]. For lack of space, we can only give a brief introduction to the latter, focusing on the main ideas. The main ingredient to our semantics is the use of ***impredicative guarded dependent type theory*** (**iGDTT**) as a sufficiently powerful metalanguage to admit both the synthetic solution of domain equations for recursively defined semantic worlds (which uses guarded recursion) *and* the definition of the store-passing monad (which uses impredicativity). In Section 4.1 we give a brief overview of **iGDTT**, and we proceed in Section 4.2 to explain the interpretation of higher-order store.

4.1 Impredicative guarded dependent type theory

Impredicative guarded dependent type theory or **iGDTT** is roughly the extension of extensional ***guarded dependent type theory*** [14,15] by additional (impredicative) universe structure. We first describe the universes in Section 4.1.1, and then briefly explain the basics of **iGDTT**'s synthetic guarded domain theory in Sections 4.1.2 and 4.1.3. Finally, we describe a simple recipe for constructing models of **iGDTT** in Section 4.1.4 from which consistency immediately follows.

4.1.1 Impredicative universes

iGDTT adds to ordinary guarded dependent type theory the following additional universe structure:

(i) We assume an ordinary hierarchy of predicative universes $\mathsf{Type}_0 : \mathsf{Type}_1 : \ldots$; when it causes no confusion, we will write Type for any appropriate Type_i.

(ii) We further assume a pair of impredicative universes $\mathsf{Set}, \mathsf{Prop} : \mathsf{Type}_0$ of small types and proof-irrelevant propositions respectively, where the latter satisfies propositional extensionality. [8] Finally, we assume that any element of Prop is also an element of Set.

The universe structure above is roughly that of the `Set` and `Prop` universes of Coq [16] underneath `Type` when the `-impredicative-set` option is activated. Impredicativity of Set means closure under dependent products $\forall_{x:A} B : \mathsf{Set}$ of "large-indexed" families $x : A \vdash B : \mathsf{Set}$ when $A : \mathsf{Type}$, and likewise for Prop. The coherent impredicative encoding of Awodey, Frey, and Speight [6] ensures that the full internal subcategory determined by Set is *reflective* in Type, and so a genuine existential $\exists_{x:A} B : \mathsf{Set}$ can be obtained by applying the reflection to the (large) dependent sum $\sum_{x:A} B : \mathsf{Type}$. On Prop these are exactly the ordinary universal and existential quantifiers — as the reflection is the "bracket type" of Awodey and Bauer [5].

In what follows, we shall let $\mathcal{U}$ stand for any of the universes of **iGDTT** described here.

4.1.2 The later modality and guarded recursion

Every universe $\mathcal{U}$ is closed under a "later modality" $\blacktriangleright : \mathcal{U} \to \mathcal{U}$ facilitating guarded recursion. The later modality also satisfies a dependently typed version of the rules of an applicative functor. Although there are many ways to present this structure, we choose to follow prior work [14,15,46] by formulating them using ***delayed substitutions*** $\delta \rightsquigarrow \Delta$, which we describe simultaneously with the rules of the later modality:

LATER FORMATION
$$\frac{\delta \rightsquigarrow \Delta \qquad \Delta \vdash A\ \mathit{type}}{\blacktriangleright[\delta].A\ \mathit{type}}$$

LATER FUNCTORIALITY
$$\frac{\delta \rightsquigarrow \Delta \qquad \Delta \vdash a : A}{\mathsf{next}[\delta].a : \blacktriangleright[\delta].A}$$

EMPTY DSUBST.
$$\frac{}{\cdot \rightsquigarrow \cdot}$$

EXTENDED DSUBST.
$$\frac{\delta \rightsquigarrow \Delta \qquad a : \blacktriangleright[\delta].A}{(\delta, x \leftarrow a) \rightsquigarrow \Delta, x : A}$$

LATER WEAKENING
$$\frac{\delta \rightsquigarrow \Delta \qquad a : \blacktriangleright[\delta].A \qquad \Delta \vdash B\ \mathit{type}}{\blacktriangleright[\delta, x \leftarrow a].B = \blacktriangleright[\delta].B}$$

NEXT WEAKENING
$$\frac{\delta \rightsquigarrow \Delta \qquad a : \blacktriangleright[\delta].A \qquad \Delta \vdash b : B}{\mathsf{next}[\delta, x \leftarrow a].b = \mathsf{next}[\delta].b}$$

LATER EXCHANGE
$$\frac{\delta \rightsquigarrow \Delta \qquad a : \blacktriangleright[\delta].A \qquad b : \blacktriangleright[\delta].B \qquad \Delta, x : A, y : B \vdash \delta' \rightsquigarrow \Delta' \qquad \Delta, x : A, y : B, \Delta' \vdash C\ \mathit{type}}{\blacktriangleright[\delta, x \leftarrow a, y \leftarrow b, \delta'].C = \blacktriangleright[\delta, y \leftarrow b, x \leftarrow a, \delta'].C}$$

NEXT EXCHANGE
$$\frac{\delta \rightsquigarrow \Delta \qquad a : \blacktriangleright[\delta].A \qquad b : \blacktriangleright[\delta].B \qquad \Delta, x : A, y : B \vdash \delta' \rightsquigarrow \Delta' \qquad \Delta, x : A, y : B, \Delta' \vdash c : C}{\mathsf{next}[\delta, x \leftarrow a, y \leftarrow b, \delta'].c = \mathsf{next}[\delta, y \leftarrow b, x \leftarrow a, \delta'].c}$$

LATER FORCE
$$\frac{\delta \rightsquigarrow \Delta \qquad \Delta \vdash a : A \qquad \Delta, x : A \vdash B\ \mathit{type}}{\blacktriangleright[\delta, x \leftarrow \mathsf{next}[\delta].a].B = \blacktriangleright[\delta].B[a/x]}$$

NEXT FORCE
$$\frac{\delta \rightsquigarrow \Delta \qquad \Delta \vdash a : A \qquad \Delta, x : A \vdash b : B}{\mathsf{next}[\delta, x \leftarrow \mathsf{next}[\delta].a].b = \mathsf{next}[\delta].b[a/x]}$$

LATER ID
$$\frac{\delta \rightsquigarrow \Delta \qquad \Delta \vdash a : A \qquad \Delta \vdash a' : A}{(\mathsf{next}[\delta].a = \mathsf{next}[\delta].a') = \blacktriangleright[\delta].(a = a')}$$

NEXT VARIABLE
$$\frac{\delta \rightsquigarrow \Delta \qquad a : \blacktriangleright[\delta].A}{\mathsf{next}[\delta, x \leftarrow a].x = a}$$

Then the ordinary later modality $\blacktriangleright : \mathcal{U} \to \mathcal{U}$ sends $A : \mathcal{U}$ to $\blacktriangleright[\cdot].A$ via the empty delayed substitution; likewise, we shall write $\mathsf{next}\, a$ for $\mathsf{next}[\cdot].a$. From these rules, we may deduce that the later modality forms a ***well-pointed endofunctor*** $\blacktriangleright : \mathcal{U} \to \mathcal{U}$ in the sense of Kelly [32], and moreover preserves cartesian products. The later modality also comes equipped with a ***Löb recursor*** for defining guarded fixed points as specified below:

Löb recursor
$$\mathsf{gfix} : (\blacktriangleright A \to A) \to A$$

Löb unfolding
$$\mathsf{gfix}\, f = f(\mathsf{next}\, (\mathsf{gfix}\, f))$$

[8] Note that Prop is not an element of Set, as this would be inconsistent [17].

4.1.3 Basic synthetic guarded domain theory

Definition 4.1 A ***guarded domain*** in $\mathcal{U}$ is defined to be an algebra for the endofunctor $\blacktriangleright : \mathcal{U} \to \mathcal{U}$, *i.e.* a type $X : \mathcal{U}$ equipped with a function $\vartheta_X : \blacktriangleright A \to A$. A homomorphism from X to Y is then given by a function $f : X \to Y$ that commutes with the algebra maps in the sense that $\vartheta_Y \circ \blacktriangleright f = f \circ \vartheta_X$. We will write $\mathcal{U}^{\blacktriangleright}$ for the category of guarded domains in $\mathcal{U}$ and their homomorphisms.

Example 4.2 The universe $\mathcal{U}$ is a guarded domain in any higher universe $\mathcal{V}$ as we may define $\vartheta_{\mathcal{U}} : \blacktriangleright\mathcal{U} \to \mathcal{U}$ to send $A : \blacktriangleright\mathcal{U}$ to the delayed type $\blacktriangleright[Z \leftarrow A].Z$ using the unary delayed substitution $(Z \leftarrow A) \rightsquigarrow Z : \mathcal{U}$.

Following Birkedal and Møgelberg [10], the Löb recursor can be used to solve domain equations by computing fixed points on the universe $\mathcal{U}$. The simplest example of a guarded domain equation is the one that defines the ***guarded lift functor*** $\mathsf{L} : \mathcal{U} \to \mathcal{U}^{\blacktriangleright}$ of Paviotti, Møgelberg, and Birkedal [40] which sends a type to the free guarded domain on that type, *i.e.* the left adjoint to the forgetful functor from guarded domains to types. The domain equation in question is $\mathsf{L}A = A + \blacktriangleright\mathsf{L}A$, which we solve using the Löb recursor on the universe together with the latter's guarded domain structure:

$$\mathsf{L}A \triangleq \mathsf{gfix}\,(\lambda X : \blacktriangleright\mathcal{U}.A + \vartheta_{\mathcal{U}}X) = A + \vartheta_{\mathcal{U}}(\mathsf{next}\,(\mathsf{L}A)) = A + \blacktriangleright[Z \leftarrow \mathsf{next}\,(\mathsf{L}A)].Z = A + \blacktriangleright\mathsf{L}A$$

The algebra structure $\vartheta_{\mathsf{L}A} : \blacktriangleright\mathsf{L}A \to \mathsf{L}A$ is given by the right-hand coproduct injection; the left-hand injection then defines the unit map $\eta : A \to \mathsf{L}A$ for the monad determined by the resulting adjunction between guarded domains and types.

Construction 4.3 (Guarded domains are lift-algebras) *Any guarded domain X is also an algebra for the monad L. As $\mathsf{L}X$ is the* free $\blacktriangleright$*-algebra on X, there is a unique homomorphism of $\blacktriangleright$-algebras $\alpha_X : \mathsf{L}X \to X$ such that $\alpha_X \circ \eta = \mathsf{id}_X$ which induces an L-algebra structure on X.*

Construction 4.4 (Family lifting for the guarded lift monad) *Let $\mathcal{U}$ be a universe, and let $A : \mathsf{Type}$ be a type. We may lift a family $\Phi : A \to \mathcal{U}$ to a family $\Phi^{\mathsf{L}} : \mathsf{L}A \to \mathcal{U}$ defined using the induced L-algebra structure on $\mathcal{U}$,* i.e. *$\Phi^{\mathsf{L}} \triangleq \alpha_{\mathcal{U}} \circ \mathsf{L}\Phi$. We will occasionally write $u \Downarrow \Phi$ to mean $\Phi^{\mathsf{L}}u$.*

*4.1.4 Consistency and models of **iGDTT***

A simple and modular recipe for constructing non-trivial models of **iGDTT** is provided by Sterling, Gratzer, and Birkedal [46], from which consistency is easily deduced.

Theorem 4.5 (S., Gratzer, and B. [46]) *Let $(\mathbb{O}, \leq, \prec)$ be a* separated intuitionistic well-founded poset [9] *in a realizability topos $\mathscr{S}$. Then internal presheaves $[\mathbb{O}^{\mathsf{op}}, \mathscr{S}]$ give a model of **iGDTT** in which:*

(i) *the predicative universes Type are modeled by the Hofmann–Streicher liftings [26,4] of the universes of (small) assemblies [34] from $\mathscr{S}$;*

(ii) *the impredicative universes $\mathsf{Prop}, \mathsf{Set} : \mathsf{Type}$ are modeled by the Hofmann–Streicher liftings of the universes of $\neg\neg$-closed propositions and of modest sets in $\mathscr{S}$ respectively;*

(iii) *the later modality $\blacktriangleright$ is computed explicitly by the limit $(\blacktriangleright A)u = \varprojlim_{v \prec u} Av$.*

Example 4.6 The simplest example of a model of **iGDTT** instantiating Theorem 4.5 is given by the standard order of the natural numbers object in Hyland's [28] effective topos **Eff**; this is exactly the "topos of trees" [11] constructed internally to **Eff**. This model can be adjusted in two orthogonal directions, by varying the underlying partial combinatory algebra and by varying the internal well-founded order.

Corollary 4.7 (S., Gratzer, and B. [46]) ***iGDTT** is consistent.*

*4.2 Denotational semantics of general store in **iGDTT***

We briefly recall the denotational semantics of general store in **iGDTT** via a *presheaf model*. We will first construct a preorder $\mathbb{W}$ of semantic worlds (representing heap layouts), and then we shall interpret *program types* as Set-valued co-presheaves on $\mathbb{W}$ and *logical types* as Type-valued co-presheaves on $\mathbb{W}$.

[9] We omit the definition of *separated intuitionistic well-founded posets* for brevity and refer the reader to Sterling, Gratzer, and Birkedal [46] for details.

Notation 4.8 *When P is a partial order and $p \leq_P q$, we will write $q_* : Ep \to Eq$ for the covariant functorial action of any functor $E : P \to \mathscr{E}$.*

4.2.1 Recursively defined semantic worlds

In this section, we will define a partial order $(\mathbb{W}, \leq)$ of semantic worlds simultaneously with the collection of semantic types by solving a guarded domain equation in Type. We will ultimately define $\mathbb{W}$ to be a kind of finite mapping of locations to types, but we must be more careful than usual because notions like "finite subtype" are somewhat sensitive when Prop is not a true subobject classifier.

Definition 4.9 Let I be a totally ordered type; a ***finite subtype*** $U \subseteq_{\mathit{fin.}} I$ is an element $|U| : \mathbb{N}$ together with a monotone injective[10] function $\sigma_U : \mathbb{N}_{<|U|} \hookrightarrow I$. We will often abuse notation by writing U to refer to the image of σ_U in I. There is a (decidable) preorder on finite subtypes given by inclusion, which is in fact a partial order because of the monotonicity of σ_U in the total order I.

Definition 4.10 Given a totally ordered type I, a ***finite mapping*** $w : I \rightharpoonup_{\mathit{fin.}} T$ is given by a finite subtype $|w| \subseteq_{\mathit{fin.}} I$ called the ***support*** together with a function $\tau_w : |w| \to T$ called the ***labeling***. Given $i \in |w|$ we shall simply write $wi : T$ for $\tau_w i$. There is a partial order on finite mappings $w : I \rightharpoonup_{\mathit{fin.}} T$ given by inclusion of supports: we say that $w \leq w'$ when $|w| \leq |w'|$ and the restriction of $\tau_{w'}$ to $|w|$ is equal to τ_w. Note that the partial order on finite mappings is not decidable unless T has decidable equality.

We can now use the notion of finite mapping to define a partial order of ***semantic worlds*** $\mathbb{W}$ simultaneously with the categories $\mathcal{S}_{\boldsymbol{p}}, \mathcal{S}_{\boldsymbol{\ell}}$ of semantic program types and semantic logical types respectively by solving the following guarded domain equation:

$$\mathbb{W} = \mathbb{N} \rightharpoonup_{\mathit{fin.}} \blacktriangleright \mathcal{S}_{\boldsymbol{p}} \qquad \mathcal{S}_{\boldsymbol{p}} = [\mathbb{W}, \mathsf{Set}] \qquad \mathcal{S}_{\boldsymbol{\ell}} = [\mathbb{W}, \mathsf{Type}]$$

In the above, we have defined a world to be a finite mapping from memory locations to delayed semantic program types, which are defined to be Set-*valued co-presheaves* on the poset of semantic worlds. Semantic logical types are defined similarly as Type-valued co-presheaves.

Observation 4.11 *Note that both $\mathcal{S}_{\boldsymbol{p}}$ and $\mathcal{S}_{\boldsymbol{\ell}}$ are guarded domains in the sense of Definition 4.1: the structure map $\vartheta_{\mathcal{S}_\iota}$ sends $A : \blacktriangleright \mathcal{S}_\iota$ to the co-presheaf $w \mapsto \blacktriangleright[Z \leftarrow A].Zw$.*

4.2.2 Semantic heaplets; total heaps and partial heaps

The semantic notion of *heap* or *memory* can be specified in terms of the more general ***heaplet distributor*** on $\mathbb{W}$. This is the distributor $\mathcal{H} : \mathbb{W}^{\mathsf{op}} \times \mathbb{W} \to \mathsf{Set}$ that classifies heaps whose layout is governed by the contravariant parameter and whose values vary in the covariant parameter.

$$\mathcal{H}(w^-, w^+) \triangleq \textstyle\prod_{l \in |w^-|} \vartheta_{\mathcal{S}_{\boldsymbol{p}}}(w^- l)\, w^+ = \prod_{l \in |w^-|} \blacktriangleright[Z \leftarrow w^- l].Zw^+$$

Using the notion of a heaplet, we can define ***partial heaps*** and ***total heaps*** and at a given world; the latter are used to interpret the state monad of $\mathbf{F}_\mu^{\mathit{ref}}$, whereas the former are used to interpret the program logic. Partial heaps will be functorial in worlds, whereas total heaps are a non-functorial derived form.

Definition 4.12 A ***partial heap*** h at a world w is given by a finite subtype $\|h\| \subseteq_{\mathit{fin.}} |w|$ together with a heaplet $\eta_h : \mathcal{H}(w_{\|h\|}, w)$. We shall write $|h| \triangleq w_{\|h\|}$ for the supporting world; given $l \in \|h\|$ we shall write hl for $\eta_h l$. Partial heaps are arranged into a functor $\mathsf{pH} : \mathbb{W} \to \mathsf{Set}$; the covariant functoriality in worlds $w \leq w'$ takes a partial heap $h : \mathsf{pH}w$ to $w'_* h \triangleq (\|h\|, w'_* \eta_h)$.

Definition 4.13 Two partial heaps are ***disjoint*** from each other when their supports do not intersect. This property is both decidable and functorial in $\mathbb{W}$, so it yields a decidable subobject of $\mathsf{pH} \times \mathsf{pH}$ in $\mathcal{S}_{\boldsymbol{\ell}}$. In the internal language of $\mathcal{S}_{\boldsymbol{\ell}}$, we will write $h \# h'$ to mean that h and h' are disjoint.

[10] We mean injective in the general intuitionistic sense: elements of the domain are equal if and only if they are identified by the function in question.

Construction 4.14 *We may define an internal* ***partial commutative monoid*** *structure on* pH *in* $\mathcal{S}_{\boldsymbol{\ell}}$*. The unit is the empty heap* $\emptyset$*, and the partial multiplication* $h_1 \cdot h_2$ *is defined when* $h_1 \# h_2$ *as follows:*

$$\|h_1 \cdot_w h_2\| \triangleq \|h_1\| \cup \|h_2\|$$
$$\eta_{h_1 \cdot_w h_2} \ell \triangleq \begin{cases} |h_1 \cdot_w h_2|_* \eta_{h_1} \ell & \text{if } \ell \in \|h_1\| \\ |h_1 \cdot_w h_2|_* \eta_{h_2} \ell & \text{if } \ell \in \|h_2\| \end{cases}$$

Definition 4.15 A ***total heap*** at a world w is a partial heap $h : \mathsf{pH}w$ such that $|h| = w$; this is not functorial in $\mathbb{W}$, but we may write total heaps as a functor $\mathsf{tH}_\bullet : |\mathbb{W}| \to \mathsf{Set}$ where $|\mathbb{W}|$ is the underlying discrete category of $\mathbb{W}$. We shall abusively write tH for the dependent sum $\sum_{w:\mathbb{W}} \mathsf{tH}_w$ that bundles a heap with its world. We will write $\mathsf{tH}_{\#w}$ to mean the type of total heaps whose support is disjoint from w.

4.2.3 Semantic domains for predicates

Here we describe the semantic domains that govern predicates and entailments; these domains will ultimately form the basis for a BI-hyperdoctrine $\mathcal{B}^\bullet$ over $\mathcal{S}_{\boldsymbol{\ell}}$, to be described later. We shall denote by $\mathsf{Prop}_{\mathbb{W}}$ the Hofmann–Streicher lifting of Prop into $\mathcal{S}_{\boldsymbol{\ell}} = [\mathbb{W}, \mathsf{Type}]$, defining $\mathcal{B}$ to be the internal poset of $\mathsf{Prop}_{\mathbb{W}}$-valued co-presheaves on the partial commutative monoid pH under its extension order:

$$\mathsf{Prop}_{\mathbb{W}} : \mathcal{S}_{\boldsymbol{\ell}} \qquad \mathcal{B} : \mathcal{S}_{\boldsymbol{\ell}}$$
$$\mathsf{Prop}_{\mathbb{W}} w \triangleq [w \downarrow \mathbb{W}, \mathsf{Prop}] \qquad \mathcal{B} \triangleq [\mathsf{pH}, \mathsf{Prop}_{\mathbb{W}}]$$

Notation 4.16 (Forcing for $\mathsf{Prop}_{\mathbb{W}}$) *For any* $X : \mathcal{S}_{\boldsymbol{\ell}}$*,* $\phi : X \to \mathsf{Prop}_{\mathbb{W}}$ *and* $w : \mathbb{W}$ *and* $x : Xw$*, we shall write* $w \Vdash \phi\, x$ *in* ***iGDTT*** *to mean that* $\phi_w x\, w$ *holds.*

Notation 4.17 (Forcing for $\mathcal{B}$) *Let* $X : \mathcal{S}_{\boldsymbol{\ell}}$ *be a semantic type; then in the internal language of* $\mathcal{S}_{\boldsymbol{\ell}}$*, for any* $\phi : \mathcal{B}^X$*,* $h : \mathsf{pH}$*, and* $x : X$*, we shall write* $h \models \phi\, x$ *to mean that* $\phi\, x\, h$ *holds.*

We note that $\mathcal{S}_{\boldsymbol{\ell}}$ inherits [38] from **iGDTT** a later modality $\blacktriangleright$ defined pointwise; it follows that $\mathsf{Prop}_{\mathbb{W}}$ is closed under a later modality $\triangleright : \mathsf{Prop}_{\mathbb{W}} \to \mathsf{Prop}_{\mathbb{W}}$ satisfying $w \Vdash \triangleright(\phi\, x) \iff \blacktriangleright(w \Vdash \phi\, x)$.

4.2.4 Interpretation of judgmental structure

We summarize the interpretation of the judgmental structure of Tulip below:

(i) Type contexts $\boxed{\Xi\ \mathit{tctx}}$ are interpreted as semantic logical types $[\![\Xi]\!] : \mathsf{Type}$.

(ii) Element contexts $\boxed{\Xi \vdash \Gamma\ \mathit{ctx}}$ are interpreted as families $[\![\Gamma]\!] : [\![\Xi]\!] \to \mathcal{S}_{\boldsymbol{\ell}}$.

(iii) Types $\boxed{\Xi \vdash A\ \mathit{type} \mathbin{@} \iota}$ are interpreted as families $[\![A]\!] : [\![\Xi]\!] \to \mathcal{S}_\iota$.

(iv) Elements $\boxed{\Xi \mid \Gamma \vdash a : A}$ are interpreted as elements $[\![a]\!] : \prod_{\xi : [\![\Xi]\!]} [\![\Gamma]\!]\xi \to [\![A]\!]\xi$.

(v) Propositions $\boxed{\Xi \mid \Gamma \vdash \phi\ \mathit{prop}}$ are interpreted as predicates $[\![\phi]\!] : \prod_{\xi : \Xi} \mathcal{B}^{[\![\Gamma]\!]\xi}$.

(vi) Entailments $\boxed{\Xi \mid \Gamma \mid \phi \vdash \psi}$ are interpreted as parameterized inequalities $\bigwedge_{\xi : [\![\Xi]\!]} [\![\phi]\!]\xi \leq_{\mathcal{B}^{[\![\Gamma]\!]\xi}} [\![\psi]\!]\xi$.

4.2.5 Recursive types, general reference types, and the monad

In our semantics, recursive types are computed using the guarded domain structure of the semantic universes $\mathcal{S}_\iota$; general reference types are defined pointwise as a subtype of the world's support; the monad is likewise defined pointwise using a combination of universal types, existential types, and the guarded lifting monad:

$$\mu : (\mathcal{S}_\iota \to \mathcal{S}_\iota) \to \mathcal{S}_\iota \qquad \mathsf{ref} : \mathcal{S}_{\boldsymbol{p}} \to \mathcal{S}_{\boldsymbol{p}} \qquad \mathsf{T} : \mathcal{S}_{\boldsymbol{p}} \to \mathcal{S}_{\boldsymbol{p}}$$
$$\mu F = \mathsf{gfix}\,(\vartheta_{\mathcal{S}_\iota} \circ \blacktriangleright F) \qquad \mathsf{ref}\, A\, w = \{\ell : |w| \mid w\ell = \mathsf{next}\, A\} \qquad \mathsf{T} A w = \bigwedge_{w' \geq w} \mathsf{tH}_{w'} \to \mathsf{L} \exists_{w'' \geq w'} \mathsf{tH}_{w''} \times A w''$$

We do not have the space to display the operations of the monad; we note, however, that T is $\mathcal{S}_{\boldsymbol{p}}$-enriched and therefore strong. It follows that the semantic type operations in this section can be used to interpret

the types of $\mathbf{F}_{\mu}^{ref}$. We show how to interpret the getter and setter for reference types in the model:

$$\mathsf{set}_A : \mathsf{ref}\, A \times A \to \mathsf{T}()$$
$$(\mathsf{set}_A)_w\,(\ell : \mathsf{ref}\, A\, w, a : A\, w)\,(w' \geq w)\,(h : \mathsf{tH}_{w'}) \triangleq \eta\,(\mathsf{pack}\,(w', h[\ell \mapsto \mathsf{next}\, w'_* a], ()))$$

$$\mathsf{get}_A : \mathsf{ref}\, A \to \mathsf{T}A$$
$$(\mathsf{get}_A)_w\,(\ell : \mathsf{ref}\, A\, w)\,(w' \geq w)\,(h : \mathsf{tH}_{w'}) \triangleq \vartheta\,(\mathsf{next}[B \leftarrow w'\ell, x \leftarrow h\ell].\,\eta\,(\mathsf{pack}\,(w', h, x)))$$

Note that set_A returns immediately in the guarded lift monad via the unit η, whereas get_A takes a single step via the $\blacktriangleright$-algebra map ϑ; this is because the heap stores its elements under the later modality, so reading from memory takes one abstract step of computation in the guarded lift monad. This is also reflected in the rule WP-GET, which allows an assumption to be under the later modality.

4.2.6 Semantics of logical types

We interpret the logical type of propositions `prop` as the internal poset $\mathcal{B} = [\mathsf{pH}, \mathsf{Prop}_{\mathbb{W}}]$. The interpretation of the remaining type connectives is standard.

4.3 Semantics of predicate connectives

We will impose enough structure on $\mathcal{B}$ such that the indexed partial order $\mathcal{B}^{\bullet} : \mathcal{S}_{\ell}^{\mathsf{op}} \to \mathbf{Poset}_{\mathsf{Type}}$ has the structure of a BI-hyperdoctrine with appropriate modalities ($\Box$, $\triangleright$) and weakest preconditions.

4.3.1 A complete BI-algebra

We will argue that $\mathcal{B}$ forms a ***complete BI-algebra*** in $\mathcal{S}_{\ell}$. Note that in this section, when we say that a partial order is *complete*, we mean that it is complete in the sense of internal category theory [29].

Lemma 4.18 *$\mathsf{Prop}_{\mathbb{W}}$ is a complete Heyting algebra in $\mathcal{S}_{\ell}$.*

Corollary 4.19 *The internal poset $\mathcal{B}$ is a complete Heyting algebra in $\mathcal{S}_{\ell}$.*

We will use ***Day's convolution*** [19,20] to construct a BI-algebra structure on $\mathcal{B} = [\mathsf{pH}, \mathsf{Prop}_{\mathbb{W}}]$. Day's convolution product is most well-known for extending monoidal structures on small categories to presheaves, but we will need the full generality of his result: the categories involved are $\mathsf{Prop}_{\mathbb{W}}$-enriched, and the structure on the base is *promonoidal* rather than *monoidal*.

Construction 4.20 ($\mathsf{Prop}_{\mathbb{W}}$-enriched promonoidal structure on a pcm) *A partial commutative monoid $M = (M, \emptyset, \cdot)$ in $\mathcal{S}_{\ell}$ can be viewed as a $\mathsf{Prop}_{\mathbb{W}}$-enriched category, because its extension order is valued in $\mathsf{Prop}_{\mathbb{W}}$. As a $\mathsf{Prop}_{\mathbb{W}}$-enriched category, M has a $\mathsf{Prop}_{\mathbb{W}}$-enriched **promonoidal structure** in the sense of Day [19,20], which essentially encodes graph of the partial multiplication operation:*

(i) *The $\mathsf{Prop}_{\mathbb{W}}$-distributor $\mathsf{mul} : M \times (M \times M)^{\mathsf{op}} \to \mathsf{Prop}_{\mathbb{W}}$ sends $(m, (n_0, n_1))$ to $\exists n_2. m = n_0 \cdot n_1 \cdot n_2$.*

(ii) *The $\mathsf{Prop}_{\mathbb{W}}$-distributor $\mathsf{unit} : M \times \mathbf{1}^{\mathsf{op}} \to \mathsf{Prop}_{\mathbb{W}}$ sends $(m, *)$ to the proposition $(m = \emptyset)$.*

(iii) *The associativity and unit isomorphisms are defined using the associativity and unit laws for the partial multiplication operation.*

Construction 4.21 (BI algebra) *We obtain a BI-algebra structure $(*, -\!\!*)$ on $\mathcal{B} = [\mathsf{pH}, \mathsf{Prop}_{\mathbb{W}}]$ by taking the Day convolution of the induced $\mathsf{Prop}_{\mathbb{W}}$-enriched promonoidal structure (Construction 4.20) on the partial commutative monoid pH, such that the separating conjunction extends the partial multiplication operation on representables. In particular, if $h \# h'$ are two disjoint partial heaps, then $\mathbf{y}h * \mathbf{y}h' = \mathbf{y}(h \cdot h')$.*

These constructions are explained in more detail for BI-algebras arising from partial commutative monoids by Bizjak and Birkedal [13].

4.3.2 Modalities: persistence and later

The persistence modality is interpreted as the map $\Box : \mathcal{B} \to \mathcal{B}$ obtained by reindexing along the constant endomap $h \mapsto \emptyset$, sending $\phi : \mathcal{B}$ to $h \mapsto \phi\,\emptyset$. The later modality $\triangleright : \mathcal{B} \to \mathcal{B}$ is given *pointwise*.

4.3.3 The points-to predicate

We interpret the points-to predicate in the generic case $[\![\alpha \mid \ell : \mathtt{ref}\, A, a : A \vdash [\ell \hookrightarrow a]\ \mathit{prop}]\!]$. In particular, we must define for each program type $A : \mathcal{S}_{\boldsymbol{p}}$ a natural transformation $[- \hookrightarrow -] : \mathsf{ref}\, A \times A \to \mathcal{B}$, which will turn out to be representable by a singleton heap, *i.e.* we may define $h \models [\ell \hookrightarrow a] \iff \{l \mapsto \mathsf{next}\, a\} \leq h$.

4.3.4 Weakest preconditions

Finally we must interpret the weakest precondition connective, which we do in the generic case of $[\![\alpha \mid \phi : \alpha \to \mathtt{prop}, u : \mathtt{T}\alpha \vdash \mathtt{wp}\, u\, \{x.\phi\, x\}\ \mathit{prop}]\!]$. This amounts to constructing for each semantic program type $A : \mathcal{S}_{\boldsymbol{p}}$ a natural transformation $\mathsf{wp}_A : \mathcal{B}^A \times \mathsf{T}A \to \mathcal{B}$. As the denotation of the state monad is defined world-by-world, so must be the interpretation of $h \models \mathsf{wp}_A\, u\, \phi$; to that end, we give the forcing clause for $w \Vdash (h \models \mathsf{wp}_A\, u\, \phi)$ in the external **iGDTT** language as follows, recalling the $\Downarrow$ notation for the predicate lifting of L from Construction 4.4:

$$
\begin{aligned}
&w \Vdash (h \models \mathsf{wp}_A\, u\, \phi) \iff \\
&\quad \forall (w' \geq w)\, (h_f : \mathsf{pH}w')\, (h_t : \mathsf{tH}_{w'})\, (h_t = h_f \cdot w'_* h). \\
&\quad u\, w'\, h_t \Downarrow \lambda p. \\
&\qquad \exists (w'' \geq |h|)\, (w_r = w'' \cdot |h_f|)\, (h' : \mathsf{tH}_{w''})\, (a : A w_r). \\
&\qquad p = \mathsf{pack}\, (w_r, (w_r)_* h_f \cdot (w_r)_* h', a) \\
&\qquad \wedge\ w_r \Vdash ((w_r)_* h' \models \phi_{w_r}\, a = \top)
\end{aligned}
$$

Although the definition is quite technical, the idea is simple. The denotation of a monadic program is a guarded-recursive process taking a heap and ultimately producing a return configuration at a larger world. In simple terms, the weakest precondition of a predicate ϕ should quantify over all frames for the starting heap and check that the process returns only configurations satisfying ϕ without disturbing the frame.

4.3.5 Explicit Kripke–Joyal translation

We have given the interpretation of our logic in a mostly abstract–categorical way; such an abstract presentation verifies all the "logical" rules of our system, but explicit computations are needed in order to verify the rules for weakest preconditions. In this section, we provide some tools to assist with these explicit computations; in Computation 4.22 we describe how to interpret each of the main connectives of the logic as a transformer of subobjects in the internal language of $\mathcal{S}_{\boldsymbol{\ell}}$.

Computation 4.22 (Kripke–Joyal translation of the $\mathcal{B}$ logic) *The action of each connective on $\mathcal{B} = [\mathsf{pH}, \mathsf{Prop}_{\mathbb{W}}]$ can be computed explicitly as a forcing clause in the Kripke–Joyal translation [35]. We omit the forcing clauses for $\top, \bot, \wedge, \vee, \exists, \triangleright$ because they are pointwise:*

$$
\begin{aligned}
h \models \phi\, x \Rightarrow \psi\, x &\iff \forall (h' \geq h).\, h' \models \phi\, x \Rightarrow h' \models \psi\, x \\
h \models \forall_Y \phi(x, -) &\iff \forall (h' \geq h)\, (y : Y).\, h' \models \phi\, (x, y) \\
h \models \Box(\phi\, x) &\iff \emptyset \models \phi\, x \\
h \models \phi\, x * \psi\, x &\iff \exists (h_1 \cdot h_2 = h).\, h_1 \models \phi\, x \wedge h_2 \models \psi\, x \\
h \models \phi\, x \mathrel{-\!\!*} \psi\, x &\iff \forall (h' \# h).\, h' \models \phi\, x \Rightarrow h \cdot h' \models \psi\, x \\
h \models [l \hookrightarrow a] &\iff \{\ell \mapsto \mathsf{next}\, a\} \leq h
\end{aligned}
$$

Computation 4.23 (Kripke–Joyal translation of the $\mathsf{Prop}_{\mathbb{W}}$ logic) *The connectives on $\mathsf{Prop}_{\mathbb{W}}$ can be further computed in terms of the ambient **iGDTT** model by another layer of Kripke–Joyal forcing:*

$$
\begin{aligned}
w \Vdash \phi\, x \Rightarrow \psi\, x &\iff \forall (w' \geq w).\, w' \Vdash \phi\, (w'_* x) \Rightarrow w' \Vdash \psi\, (w'_* x) \\
w \Vdash \forall_Y \phi(x, -) &\iff \forall (w' \geq w)\, (y : Y w').\, w' \Vdash \phi\, (w'_* x, y)
\end{aligned}
$$

4.4 Soundness results

The following results are stated internally to **iGDTT**.

Theorem 4.24 (Soundness) *If $\Xi \mid \Gamma \mid \phi \vdash \psi$ is derivable in Tulip, then for any $\xi : [\![\Xi\ \mathit{tctx}]\!]$ it holds that $[\![\Xi \mid \Gamma \vdash \phi\ \mathit{prop}]\!]\xi \leq [\![\Xi \mid \Gamma \vdash \psi\ \mathit{prop}]\!]\xi$ in $\mathcal{B}^{[\![\Xi \vdash \Gamma\ \mathit{ctx}]\!]\xi}$.*

Proof. Since $\mathcal{B}^\bullet$ is a BI-hyperdoctrine, all the specified rules of higher-order separation logic are valid [8], and the rules for the modalities follow similarly. What remains is to verify rules for weakest preconditions; we show just one illustrative case in Lemma 4.25. □

Lemma 4.25 *The following weakest precondition law for* `get` *is valid:*

$$\alpha \mid \phi : \alpha \to \mathtt{prop}, \ell : \mathtt{ref}\,\alpha, a : \alpha \mid [\ell \hookrightarrow a] * \triangleright([\ell \hookrightarrow a] \mathrel{-\!\!*} \phi\, a) \vdash \mathtt{wp}\,(\mathtt{get}\,\ell)\,\{x.\phi\, x\}$$

Proof. We will follow the Kripke–Joyal unfolding of the separation logic from Computation 4.22. Fixing $A : \mathcal{S}_{\boldsymbol{p}}$, we must prove the following:

$$\begin{array}{l}\forall\,(h_1 \# h_2 : \mathsf{pH})\,(\phi : A \to \mathcal{B})\,(\ell : \mathsf{ref}\,A)\,(a : A).\\ \quad (\{\ell \mapsto \mathsf{next}\,a\} \le h_1) \wedge \triangleright(\forall (h_3 \# h_2).\ \{\ell \mapsto a\} \le h_3 \Rightarrow h_2 \cdot h_3 \models \phi\, a)\\ \quad \Rightarrow h_1 \cdot h_2 \models \mathsf{wp}_A\,(\mathsf{get}_A\,\ell)\,\phi\end{array}$$

At this point, the only way to unfold further is to pass through a second Kripke–Joyal translation (Computation 4.23), where the indexing comes from $\mathbb{W}$ rather than pH. What follows, therefore, will be in ambient **iGDTT** language rather than the internal language of $\mathcal{S}_{\boldsymbol{\ell}}$; in particular, we assume the following:

$$\begin{array}{l}(w : \mathbb{W})\,(h_1 \# h_2 : \mathsf{pH}w)\,(\phi : \mathcal{B}^A w)\,(\ell : \mathsf{ref}\,A\,w)\,(a : A\,w) \text{ such that:}\\ \quad H_1 : \{\ell \mapsto \mathsf{next}\,a\} \le h_1\\ \quad H_2 : \blacktriangleright\big(\forall (w' \ge w)\,(h_3 \# w'_* h_2 : \mathsf{pH}w').\ \{\ell \mapsto \mathsf{next}\,w'_* a\} \le h_3 \Rightarrow w' \Vdash (w'_* h_2 \cdot h_3 \models \phi_{w'}\,(w'_* a) = \top)\big)\end{array}$$

Our goal is to prove $w \Vdash (h_1 \cdot h_2 \models \mathsf{wp}_A\,(\mathsf{get}_A\,\ell)\,\phi)$; we fix a world $w' \ge w$, a frame $h_f : \mathsf{pH}w'$ and a total heap $h_t : \mathsf{tH}_{w'}$ such that $h_t = h_f \cdot w'_* h_1 \cdot w'_* h_2$ to prove the following:

$$\begin{array}{l}\mathsf{get}_A\,\ell\,w'\,h_t \Downarrow \lambda p.\\ \quad \exists (w'' \ge |h_1 \cdot h_2|)\,(w_r = w'' \cdot |h_f|)\,(h' : \mathsf{tH}_{w''})\,(a : A w_r).\\ \quad p = \mathsf{pack}\,(w_r, (w_r)_* h_f \cdot (w_r)_* h', a)\\ \quad \wedge\ w_r \Vdash ((w_r)_* h' \models \phi_{w_r}\, a = \top)\end{array}$$

Our assumption H_1 guarantees that $\mathsf{get}_A\,\ell\,w'\,h_t$ is equal to $\vartheta\,(\mathsf{next}\,(\eta\,(\mathsf{pack}\,(w', h_t, w'_* a))))$. Therefore, our goal reduces to the following by definition of the L–predicate lifting:

$$\begin{array}{l}\blacktriangleright \exists (w'' \ge |h_1 \cdot h_2|)\,(w_r = w'' \cdot |h_f|)\,(h' : \mathsf{tH}_{w''})\,(x : A w_r).\\ \quad \mathsf{pack}\,(w', h_t, w'_* a) = \mathsf{pack}\,(w_r, (w_r)_* h_f \cdot (w_r)_* h', x)\\ \quad \wedge\ w_r \Vdash ((w_r)_* h' \models \phi_{w_r}\, x = \top)\end{array}$$

Going under the later modality in the goal, we discharge the corresponding modality from H_2. Then we instantiate $w'' \triangleq |h_1| \cdot |h_2|$ and $w_r \triangleq w'$ and $h' \triangleq w''_* h_1 \cdot w''_* h_2$ and $x \triangleq w'_* a$. Our remaining goal is $w' \Vdash (w'_* h_1 \cdot w'_* h_2 \models \phi_{w'}\,(w'_* a) = \top)$, which follows by instantiating H_2 with $w' \triangleq w'$ and $h_3 \triangleq w'_* h_1$. □

Corollary 4.26 (Consistency) *It is not the case that* $\top \le \bot$ *in* $\mathcal{B}^{[\![\cdot]\!]\xi}$ *for any* $\xi : [\![\Xi]\!]$.

5 Conclusions; related and future work

5.1 Comparison with operationally-based program logics

We have contributed a program logic Tulip over the equational theory of polymorphic, general recursive, higher-order stateful programs by building on the recent denotational semantics of general references and polymorphism of Sterling *et al.* [46], adapting many ideas that were first developed in the context of operationally based program logics. Prior works on the operational side such as Iris [30], the Verified Software Toolchain [1], and TaDA [18] have reached great heights of expressivity, incorporating constructs such as invariants and higher-order ghost state which are critical for reasoning about concurrent programs. For the sake of simplicity, we have restricted our attention to a *fixed* notion of resource (partial heaps), but we hope in the future to adapt more sophisticated constructs including higher-order ghost state, *etc.* to reach parity with existing operationally-based program logics.

5.2 Other denotationally-based program logics for state

The model of Sterling *et al.* is not the only denotational semantics of state. First-order store, both local and global, is well-represented in the literature [37,36,42,45]; there is also Levy's model of non-polymorphic higher-order store [33], and notably, a model of ***local*** *full ground store* by Kammar *et al.* [31]. Polzer and Goncharov [43] have constructed a BI-hyperdoctrine over the denotational semantics of Kammar *et al.*, and our own work is much in the spirit of theirs. However, there is an apparent mismatch between the semantics of *local store* and the model of bunched implications over it, which has impeded *op. cit.* from developing a full program logic with an interpretation of weakest preconditions.

5.3 Future perspectives

One of the methodological questions raised by our work is where, exactly, to draw the line between equational reasoning and logical reasoning. For instance, conventional operationally-based program logics do not use equational reasoning at all: our logic, in contrast, allows some equational reasoning but it is limited by the intensionality of Sterling *et al.*'s model. One possible direction for future work is to attempt to make the model itself less intensional, either by enhancing the semantic worlds [22] or by improving the interpretation of the state monad [31].

Another question is how Tulip can be implemented in a practical tool. Currently, definitional and logical equality coincide due to equality reflection, which leads the former to become undecidable. An implementation will therefore require a more refined account of the interaction between definitional and logical equality.

Acknowledgments

We wish to thank Daniel Gratzer for many helpful discussions. This research was supported in part by a Villum Investigator grant (no. 25804), Center for Basic Research in Program Verification (CPV), from the VILLUM Foundation, and in part by the European Union under the Marie Skłodowska-Curie Actions Postdoctoral Fellowship project *TypeSynth: synthetic methods in program verification*. Views and opinions expressed are however those of the authors only and do not necessarily reflect those of the European Union or the European Commission. Neither the European Union nor the granting authority can be held responsible for them.

References

[1] Appel, A. W., *Verified software toolchain*, in: *Proceedings of the 20th European Conference on Programming Languages and Systems: Part of the Joint European Conferences on Theory and Practice of Software*, ESOP'11/ETAPS'11, pages 1–17, Springer-Verlag, Saarbrücken, Germany (2011), ISBN 978-3-642-19717-8.

[2] Appel, A. W. and D. McAllester, *An indexed model of recursive types for foundational proof-carrying code*, ACM Transactions on Programming Languages and Systems **23**, pages 657–683 (2001), ISSN 0164-0925. https://doi.org/10.1145/504709.504712

[3] Awodey, S., *Natural models of homotopy type theory*, Mathematical Structures in Computer Science **28**, pages 241–286 (2018). 1406.3219. https://doi.org/10.1017/S0960129516000268

[4] Awodey, S., *On Hofmann–Streicher universes* (2022). Unpublished manuscript. https://doi.org/10.48550/ARXIV.2205.10917

[5] Awodey, S. and A. Bauer, *Propositions As [Types]*, Journal of Logic and Computation **14**, pages 447–471 (2004), ISSN 0955-792X. https://doi.org/10.1093/logcom/14.4.447

[6] Awodey, S., J. Frey and S. Speight, *Impredicative encodings of (higher) inductive types*, in: *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 76–85, Association for Computing Machinery, Oxford, United Kingdom (2018), ISBN 978-1-4503-5583-4. https://doi.org/10.1145/3209108.3209130

[7] Bénabou, J., *Distributors at work* (2000). Notes by Thomas Streicher from lectures given at TU Darmstadt.

[8] Biering, B., L. Birkedal and N. Torp-Smith, *BI-hyperdoctrines, higher-order separation logic, and abstraction*, ACM Transactions on Programming Languages and Systems **29** (2007), ISSN 0164-0925.

[9] Birkedal, L. and A. Bizjak, *Lecture notes on Iris: Higher-order concurrent separation logic* (2022). https://iris-project.org/tutorial-material.html

[10] Birkedal, L. and R. E. Møgelberg, *Intensional type theory with guarded recursive types qua fixed points on universes*, in: *Proceedings of the 2013 28th Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 213–222, IEEE Computer Society, Washington, DC, USA (2013), ISBN 978-0-7695-5020-6, ISSN 1043-6871. https://doi.org/10.1109/LICS.2013.27

[11] Birkedal, L., R. E. Møgelberg, J. Schwinghammer and K. Støvring, *First steps in synthetic guarded domain theory: Step-indexing in the topos of trees*, in: *Proceedings of the 2011 IEEE 26th Annual Symposium on Logic in Computer Science*, pages 55–64, IEEE Computer Society, Washington, DC, USA (2011), ISBN 978-0-7695-4412-0. 1208.3596. https://doi.org/10.1109/LICS.2011.16

[12] Birkedal, L., K. Støvring and J. Thamsborg, *Realisability semantics of parametric polymorphism, general references and recursive types*, Mathematical Structures in Computer Science **20**, pages 655–703 (2010). https://doi.org/10.1017/S0960129510000162

[13] Bizjak, A. and L. Birkedal, *On models of higher-order separation logic*, Electronic Notes in Theoretical Computer Science **336**, pages 57–78 (2018). https://doi.org/10.1016/j.entcs.2018.03.016

[14] Bizjak, A., H. B. Grathwohl, R. Clouston, R. E. Møgelberg and L. Birkedal, *Guarded dependent type theory with coinductive types*, in: B. Jacobs and C. Löding, editors, *Foundations of Software Science and Computation Structures: 19th International Conference, FOSSACS 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2–8, 2016, Proceedings*, pages 20–35, Springer Berlin Heidelberg, Berlin, Heidelberg (2016), ISBN 978-3-662-49630-5. 1601.01586. https://doi.org/10.1007/978-3-662-49630-5_2

[15] Bizjak, A. and R. E. Møgelberg, *Denotational semantics for guarded dependent type theory*, Mathematical Structures in Computer Science **30**, pages 342–378 (2020). https://doi.org/10.1017/S0960129520000080

[16] Coq Development Team, T., *The Coq Proof Assistant Reference Manual* (2016).

[17] Coquand, T., *An Analysis of Girard's Paradox*, in: *Proceedings of the First Symposium on Logic in Computer Science*, pages 227–236, IEEE Computer Society (1986), ISBN 0-8186-0720-3.

[18] da Rocha Pinto, P., T. Dinsdale-Young and P. Gardner, *TaDA: A Logic for Time and Data Abstraction*, in: R. E. Jones, editor, *Proceedings of the 28th European Conference on Object-Oriented Programming (ECOOP'14)*, volume 8586 of *Lecture Notes in Computer Science*, pages 207–231, Springer (2014). https://doi.org/10.1007/978-3-662-44202-9_9

[19] Day, B., *On closed categories of functors*, in: S. MacLane, H. Applegate, M. Barr, B. Day, E. Dubuc, Phreilambud, A. Pultr, R. Street, M. Tierney and S. Swierczkowski, editors, *Reports of the Midwest Category Seminar IV*, pages 1–38, Springer Berlin Heidelberg, Berlin, Heidelberg (1970), ISBN 978-3-540-36292-0.

[20] Day, B., *An embedding theorem for closed categories*, in: G. M. Kelly, editor, *Proceedings Sydney Category Theory Seminar 1972/1973*, volume 420 of *Lecture Notes in Mathematics*, pages 55–64, Springer (1974). https://doi.org/10.1007/BFb0063096

[21] Dreyer, D., A. Ahmed and L. Birkedal, *Logical step-indexed logical relations*, in: *2009 24th Annual IEEE Symposium on Logic In Computer Science*, pages 71–80 (2009), ISSN 1043-6871. https://doi.org/10.1109/LICS.2009.34

[22] Dreyer, D., G. Neis, A. Rossberg and L. Birkedal, *A relational modal logic for higher-order stateful ADTs*, in: *Proceedings of the 37th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '10, pages 185–198, Association for Computing Machinery, Madrid, Spain (2010), ISBN 978-1-60558-479-9. https://doi.org/10.1145/1706299.1706323

[23] Gordon, M., *From LCF to HOL: A short history*, in: G. Plotkin, C. Stirling and M. Tofte, editors, *Proof, Language, and Interaction*, pages 169–185, MIT Press, Cambridge, MA, USA (2000), ISBN 0-262-16188-5.

[24] Gordon, M., R. Milner and C. Wadsworth, *Edinburgh LCF: A Mechanized Logic of Computation*, volume 78 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, Heidelberg (1979).

[25] Harper, R., *Practical Foundations for Programming Languages*, Cambridge University Press, New York, NY, USA, second edition (2016).

[26] Hofmann, M. and T. Streicher, *Lifting Grothendieck universes* (1997). Unpublished note. https://www2.mathematik.tu-darmstadt.de/~streicher/NOTES/lift.pdf

[27] Huffman, B., *HOLCF '11: A Definitional Domain Theory for Verifying Functional Programs*, Ph.D. thesis, Portland State University (2012).

[28] Hyland, J. M. E., *The effective topos*, in: A. S. Troelstra and D. V. Dalen, editors, *The L.E.J. Brouwer Centenary Symposium*, pages 165–216, North Holland Publishing Company (1982).

[29] Jacobs, B., *Categorical Logic and Type Theory*, number 141 in Studies in Logic and the Foundations of Mathematics, North Holland, Amsterdam (1999).

[30] Jung, R., R. Krebbers, J.-H. Jourdan, A. Bizjak, L. Birkedal and D. Dreyer, *Iris from the ground up: A modular foundation for higher-order concurrent separation logic*, Journal of Functional Programming **28**, page e20 (2018). https://doi.org/10.1017/S0956796818000151

[31] Kammar, O., P. B. Levy, S. K. Moss and S. Staton, *A monad for full ground reference cells*, in: *Proceedings of the 32nd Annual ACM/IEEE Symposium on Logic in Computer Science*, IEEE Press, Reykjavik, Iceland (2017), ISBN 978-1-5090-3018-7. https://doi.org/10.1109/LICS.2017.8005109

[32] Kelly, G. M., *A unified treatment of transfinite constructions for free algebras, free monoids, colimits, associated sheaves, and so on*, Bulletin of the Australian Mathematical Society **22**, pages 1–83 (1980). https://doi.org/10.1017/S0004972700006353

[33] Levy, P. B., *Possible world semantics for general storage in call-by-value*, pages 232–246 (2002), ISBN 978-3-540-44240-0. https://doi.org/10.1007/3-540-45793-3_16

[34] Luo, Z., *Computation and Reasoning: A Type Theory for Computer Science*, volume 11 of *International Series of Monographs on Computer Science*, Oxford Science Publications (1994).

[35] Mac Lane, S. and I. Moerdijk, *Sheaves in geometry and logic: a first introduction to topos theory*, Universitext, Springer, New York (1992), ISBN 0-387-97710-4.

[36] Moggi, E., *Notions of computation and monads*, Information and Computation **93**, pages 55–92 (1991), ISSN 0890-5401. Selections from 1989 IEEE Symposium on Logic in Computer Science. https://doi.org/10.1016/0890-5401(91)90052-4

[37] Oles, F. J., *Type Algebras, Functor Categories and Block Structure*, pages 543–573, Cambridge University Press, USA (1986).

[38] Palombi, D. and J. Sterling, *Classifying topoi in synthetic guarded domain theory*, Electronic Notes in Theoretical Informatics and Computer Science **Volume 1 - Proceedings of MFPS XXXVIII** (2023). https://doi.org/10.46298/entics.10323

[39] Paulson, L. C., *Logic and computation : interactive proof with Cambridge LCF*, Cambridge tracts in theoretical computer science, Cambridge University Press, Cambridge, New York, Port Chester (1987), ISBN 0-521-34632-0. Autre tirage : 1990 (br.).

[40] Paviotti, M., R. E. Møgelberg and L. Birkedal, *A model of PCF in Guarded Type Theory*, Electronic Notes in Theoretical Computer Science **319**, pages 333–349 (2015), ISSN 1571-0661. The 31st Conference on the Mathematical Foundations of Programming Semantics (MFPS XXXI). https://doi.org/10.1016/j.entcs.2015.12.020

[41] Plotkin, G. D., *LCF considered as a programming language*, Theoretical Computer Science **5**, pages 223–255 (1977), ISSN 0304-3975. https://doi.org/10.1016/0304-3975(77)90044-5

[42] Plotkin, G. D. and J. Power, *Notions of computation determine monads*, in: *Proceedings of the 5th International Conference on Foundations of Software Science and Computation Structures*, pages 342–356, Springer-Verlag, Berlin, Heidelberg (2002), ISBN 3-540-43366-X.

[43] Polzer, M. and S. Goncharov, *Local local reasoning: A bi-hyperdoctrine for full ground store*, in: J. Goubault-Larrecq and B. König, editors, *Foundations of Software Science and Computation Structures*, pages 542–561, Springer International Publishing, Cham (2020), ISBN 978-3-030-45231-5.

[44] Scott, D. S., *A type-theoretical alternative to ISWIM, CUCH, OWHY*, Theoretical Computer Science **121**, pages 411–440 (1993), ISSN 0304-3975. https://doi.org/10.1016/0304-3975(93)90095-B

[45] Staton, S., *Completeness for algebraic theories of local state*, in: L. Ong, editor, *Foundations of Software Science and Computational Structures*, pages 48–63, Springer Berlin Heidelberg, Berlin, Heidelberg (2010), ISBN 978-3-642-12032-9.

[46] Sterling, J., D. Gratzer and L. Birkedal, *Denotational semantics of general store and polymorphism* (2022). Unpublished manuscript. https://doi.org/10.48550/arXiv.2210.02169

[47] Xia, L., Y. Zakowski, P. He, C.-K. Hur, G. Malecha, B. C. Pierce and S. Zdancewic, *Interaction trees: Representing recursive and impure programs in Coq*, Proceedings of the ACM on Programming Languages **4** (2019). https://doi.org/10.1145/3371119

A Omitted rules

A.1 Equational theory of a strong monad

$$\frac{\Xi \mid \Gamma \vdash u : A \qquad \Xi \mid \Gamma, x : A \vdash v : \mathtt{T}\, B}{\Xi \mid \Gamma \vdash x \leftarrow \mathtt{ret}\, u; v \equiv v[u/x] : \mathtt{T}\, B} \qquad \frac{\Xi \mid \Gamma \vdash u : \mathtt{T}\, A}{\Xi \mid \Gamma \vdash x \leftarrow u; \mathtt{ret}\, x \equiv u : \mathtt{T}\, A}$$

$$\frac{\Xi \mid \Gamma \vdash u : \mathtt{T}\, A \qquad \Xi \mid \Gamma, x : A \vdash v : \mathtt{T}\, B \qquad \Xi \mid \Gamma, y : B \vdash v : \mathtt{T}\, C}{\Xi \mid \Gamma \vdash y \leftarrow (x \leftarrow u; v); w \equiv x \leftarrow u; y \leftarrow v; w : \mathtt{T}\, C}$$

A.2 Equational theory of universal and existential types

$$\frac{\Xi, \alpha \mid \Gamma \vdash u : A \qquad \Xi \vdash B \ \mathit{type} \mathbin{@} \boldsymbol{p}}{\Xi \mid \Gamma \vdash (\lambda\alpha.u)B \equiv u[B/\alpha] : A[B/\alpha]} \qquad \frac{\Xi \mid \Gamma \vdash u : \forall\alpha.A}{\Xi \mid \Gamma \vdash u \equiv \Lambda\alpha.u \cdot \alpha : \forall\alpha.A}$$

$$\frac{\Xi \vdash B, C \ \mathit{type} \mathbin{@} \boldsymbol{p} \qquad \Xi, \alpha \vdash A \ \mathit{type} \mathbin{@} \boldsymbol{p} \qquad \Xi \mid \Gamma \vdash u : A[B/\alpha] \qquad \Xi, \alpha \mid \Gamma, x : A \vdash v : C}{\Xi \mid \Gamma \vdash \mathtt{let\ pack}\,(\alpha, x) = \mathtt{pack}\,(B, u)\ \mathtt{in}\ v \equiv v[B, u/\alpha, x] : C}$$

$$\frac{\Xi \vdash C \ \mathit{type} \mathbin{@} \boldsymbol{p} \qquad \Xi \mid \Gamma, z : \exists\alpha.A \vdash u : C \qquad \Xi \mid \Gamma \vdash v : \exists\alpha.A}{\Xi \mid \Gamma \vdash u[v/z] \equiv \mathtt{let\ pack}\,(\alpha, x) = v\ \mathtt{in}\ u[\mathtt{pack}\,(\alpha, x)/z] : C}$$

B Omitted proofs

Lemma 4.18. $\mathsf{Prop}_{\mathbb{W}}$ *is a complete Heyting algebra in* $\mathcal{S}_{\boldsymbol{\ell}}$.

Proof. The simplest way to see this is to embed $\mathcal{S}_{\boldsymbol{\ell}} = [\mathbb{W}, \mathsf{Type}_0]$ into the larger co-presheaf category $\mathscr{E} = [\mathbb{W}, \mathsf{Type}_1]$, where we have a Hofmann–Streicher lifting $\mathcal{V}$ of Type_0 whose global points correspond to $\mathcal{S}_{\boldsymbol{\ell}}$. Then the *completeness* can be expressed internally in terms of quantification over $\mathcal{V}$; this internal quantification automatically satisfies the appropriate Beck–Chevalley conditions when externalized. In particular, that $\mathsf{Prop}_{\mathbb{W}}$ has internal products then amounts to the following pullback square existing [3]:

$$\begin{array}{ccc} \mathcal{V} & \xrightarrow{\quad !_{\mathcal{V}} \quad} & \mathbf{1} \\ {\scriptstyle A \mapsto (A, \lambda\_.\top)}\downarrow & \lrcorner & \downarrow{\scriptstyle \top} \\ \sum_{A:\mathcal{V}}(A \to \mathsf{Prop}_{\mathbb{W}}) & \xrightarrow[(\forall)]{} & \mathsf{Prop}_{\mathbb{W}} \end{array}$$

The lower map can be be constructed explicitly using the internal completeness of Prop in Type. The rest of the Heyting algebra structure is inherited from Prop à la Kripke semantics over $\mathbb{W}$. □

Corollary 4.26. *It is not the case that* $\top \leq \bot$ *in* $\mathcal{B}^{[\![\cdot]\!]\xi}$ *for any* $\xi : [\![\Xi]\!]$.

Proof. The ordering on $\mathcal{B}^{[\![\cdot]\!]\xi}$ is defined pointwise in relative to ordering on $\mathcal{B}$, and its ordering is in turn defined pointwise relative to the ordering on Prop. Similarly, $\top$ and $\bot$ in $\mathcal{B}^{[\![\cdot]\!]\xi}$ are defined pointwise relative to $\top$ and $\bot$ in $\mathcal{B}$, which in turn are defined pointwise relative to $\top$ and $\bot$ in Prop. Since $[\![\cdot]\!]\xi$ is the terminal object, and thus in particular is inhabited, this implies that if $\top \leq \bot$ in $\mathcal{B}^{[\![\cdot]\!]\xi}$, we also have $\top \leq \bot$ in Prop. We thus conclude $\top \not\leq \bot$ in $\mathcal{B}^{[\![\cdot]\!]\xi}$. □