Proofs as stateful programs: A first-order logic with abstract Hoare triples, and an interpretation into an imperative language

Thomas Powell

July 29, 2022

Abstract

We introduce an extension of first-order logic that comes equipped with additional predicates for reasoning about an abstract state. Sequents in the logic comprise a main formula together with pre- and postconditions in the style of Hoare logic, and the axioms and rules of the logic ensure that the assertions about the state compose in the correct way. The main result of the paper is a realizability interpretation of our logic that extracts programs into a mixed functional/imperative language. All programs expressible in this language act on the state in a sequential manner, and we make this intuition precise by interpreting them in a semantic metatheory using the state monad. Our basic framework is very general, and can be instantiated and extended in a variety of different ways. We conclude by outlining several directions for future work.

1 Introduction

The Curry-Howard correspondence lies at the heart of theoretical computer science. Over the years, a multitude of different techniques for extracting programs from proofs have been developed, the majority of which translate formal proof systems into lambda calculi. As such, programs extracted from proofs are typically conceived as pure functional programs.

Everyday programmers, on the other hand, often think and write in an imperative paradigm, in terms of instructions that change some underlying global state. This is reinforced by the fact that many of the most popular programming languages, including C and Python, lean towards this style. Imperative programs are nevertheless highly complex from a mathematical perspective, and while systems such as Hoare logic [13] or separation logic [25] have been designed to reason about them, the formal extraction of imperative programs from proofs has received comparatively little attention.

In this paper, we propose a new idea in this direction, developing a formal system SL that enriches ordinary first-order logic with Hoare triples for reasoning about an abstract global state. Sequents will have the form $\Gamma \vdash \{\alpha \cdot A \cdot \beta\}$, where A is a formula and α, β assertions about the state, and proofs in the logic will include both ordinary introduction and elimination rules for predicate logic, together with special rules for reasoning about the state. We then a construct a stateful realizability interpretation (based on Kreisel's modified realizability [15]) that relates formulas in SL to terms in a mixed functional/imperative language ST. Our main result is a soundness theorem, which confirms that whenever a formula is provable in SL, we can extract a corresponding stateful realizing term in ST.

We are not the first to adapt traditional methods to extract imperative programs: A major achievement in this direction, for example, is the monograph [20], which sets up a variant of intuitionistic Hoare logic alongside a realizability translation into a standard imperative language. Other relevant examples include [7, 11, 17, 28]. However, these and almost all other prior work in this direction tend to focus on formal verification, with an eye towards using proof interpretations as a method for the synthesis of correct-by-construction software. In concrete terms, this means that the formal systems tend to be quite detailed and oriented towards program analysis, while the starting point is typically a program for which we want to

construct a verification proof, rather than a proof from which we hope to extract a potentially unfamiliar program.

Our approach, on the other hand, is much more abstract, with an emphasis on potential applications in logic and proof theory. Our basic system SL makes almost no assumptions about the structure of the state and what we are allowed to do with it. Rather, we focus on producing a general framework for reasoning about 'stateful formulas', which can then be instantiated with additional axioms to model concrete scenarios. The simplicity and generality of our framework is its most important feature, and we consider this work to be a first step towards a number of potentially interesting applications. For this reason, we include an extended section towards the end where we sketch out, in some detail, concrete ways in which our logic and interpretation could be used and expanded, including the computational semantics of proofs and probabilistic logic.

We take ideas from three main sources. The first is a case study of Berger et al. [5], in which a realizability interpretation is used to extract a version of in-place quicksort, and where the imperative nature of the extracted program is presented in a semantic way using the state monad. While their program behaves imperatively "by-chance", terms extracted from our logic are forced to be imperative, and thus our framework offers one potential solution to their open problem of designing a proof calculus which only yields imperative programs. Our second source of inspiration is the thesis of Birolo [6], where a general monadic realizability interpretation is defined and then used to give an alternative, semantic presentation of learning-based interactive realizability [2, 3]. However, our work goes beyond this in that it also involves a monadic extension of the target logic, whereas Birolo's applies to standard first-order logic. Finally, a number of ideas are taken from the author's previous work [22] on extracting stateful programs using the Dialectica interpretation. While there the state is used in a very specific and restricted way, unlike our more general presentation here, we use an analogous call-by-value monadic translation on terms.

It is important to stress that we do not claim that our work represents an optimal or complete method for extracting imperative programs from proofs, nor do we claim that it is superior to alternative methods, including the aforementioned works in the direction of verification, or, for instance, techniques based on Krivine's classical realizability [16], which could be viewed as imperative in nature. We simply offer what we consider to be a new and interesting perspective that emphasises abstraction and simplicity, and propose that our framework could prove valuable in a number of different contexts.

Overview of the paper. The main technical work that follows involves the design of three different systems, along with the realizability interpretation that connects them, namely:

- A novel extension SL of predicate logic with abstract Hoare triples, which can be extended with additional axioms for characterising the state (Section 2).
- A standard calculus ST for lambda terms with imperative commands, which can again be extended with additional constants for interacting with the state (Section 3).
- A metalanguage S^ω into which both SL and ST can be embedded (Section 4), which is used to formulate the realizability relation and prove its soundness (Section 5).

Concrete examples are given, and potential applications surveyed in Section 6.

2 The system SL: First-order logic with state

We begin by introducing our target theory SL from which stateful programs will be extracted. This is an extension of ordinary first-order logic in the sense that the latter can always be embedded into SL (we will make this precise in Proposition 2.1 below). Ultimately, we are interested not so much in SL on its own, but in theories of the form $\mathrm{SL} + \Delta_H + \Delta_S$, where Δ_H and Δ_S are collections of (respectively non-computational and computational) axioms that together characterise the state. Examples will be provided below, and an extended discussion of how $\mathrm{SL} + \Delta_H + \Delta_S$ could potentially be used is given in Section 6.

2.1 Intuitionistic first-order logic

Before defining SL, we give a standard presentation first-order intuitionistic predicate logic PL, which serves as an opportunity to fix our basic style of formal reasoning. The language of PL consists of the logical constants $\wedge, \vee, \Rightarrow, \forall, \exists, \top, \bot$, variables $x, y, z, \ldots$, along with function symbols $f, g, h, \ldots$ and predicate symbols $P, Q, R, \ldots$, each with a fixed arity. We assume the existence of at least one constant c. Terms are built from variables and function symbols as usual, and formulas are built from prime formulas $P(t_1, \ldots, t_n)$, $\top$ and $\bot$ using the logical constants. We use the usual abbreviation $\neg A :\equiv A \Rightarrow \bot$. We work in a sequent style natural deduction calculus, where sequents have the form $\Gamma \vdash_I A$ for some context Γ and formula A, and a context is a set of labelled assumptions of the form $A_1^{u_1}, \ldots, A_n^{u_n}$ for pairwise distinct labels u_i. The axioms and rules of PL are outlined in Appendix A.

2.2 Stateful first-order logic

We now define our new logical system SL, which is an extension of ordinary first-order logic with new state propositions. To be more precise, we extend the language of PL with a ternary operation $\{- \cdot - \cdot -\}$, together with special state predicate symbols $p, q, r, \ldots$, which also have a fixed arity. Terms of SL are the same as those of PL. On the other hand, there are two kinds of formulas in SL: state formulas and main formulas. A *state formula* is defined using state predicate symbols and propositional connectives as follows:

- $\top$ and $\bot$ are state formulas,
- if p a state predicate symbol of arity n and $t_1, \ldots, t_n$ are terms, then $p(t_1, \ldots, t_n)$ is a state formula,
- if α, β are state formulas, so are $\alpha \wedge \beta$, $\alpha \vee \beta$, $\alpha \Rightarrow \beta$.

A *main formula* (or just *formula*) of SL is now defined as:

- $\top$ and $\bot$ are formulas,
- if P is an ordinary predicate symbol of arity n and $t_1, \ldots, t_n$ are terms, then $P(t_1, \ldots, t_n)$ is a formula,
- if A, B are formulas, so are $A \wedge B$, $A \vee B$ and $\exists x A$,
- if A, B are formulas and α, β *state* formulas, then $A \Rightarrow \{\alpha \cdot B \cdot \beta\}$ and $\forall x \, \{\alpha \cdot A \cdot \beta\}$ are formulas.

The notions of free and bound variables, along with substitution $\alpha[t/x]$ and $A[t/x]$ can be easily defined for both state and main formulas.

Analogous to the construction of formulas, our basic proof system uses the auxiliary notion of a *state proof* in order to define a main proof. A *state sequent* has the form $\Gamma \vdash_H \alpha$ where α is a state formula and Γ a set of labelled state formulas. A proof of $\Gamma \vdash_H \alpha$ in SL is built from the axioms and rules of *classical propositional logic* i.e. the propositional axioms and rules as set out in Appendix A plus the law of excluded middle $\Gamma \vdash_H \alpha \vee \neg\alpha$, together with a set Δ_H of as yet unspecified state axioms of the form $\Gamma \vdash_H \alpha$.

A main sequent of SL has the form $\Gamma \vdash_S \{\alpha \cdot A \cdot \beta\}$, where A is a formula and α, β state formulas, and Γ is a set of labelled main formulas. A proof of $\Gamma \vdash_S \{\alpha \cdot A \cdot \beta\}$ in SL uses the axioms and rules given in Figure 1, together with a set Δ_S of additional axioms.

We now make precise what we mean when we characterise SL as an extension of standard first-order logic. The following is provable with an easy induction over derivations in PL:

Proposition 2.1. *For any formula A of* PL*, define the main formula A_S of* SL *by*

- $Q_S := Q$ *for Q atomic,*
- $(A \wedge B)_S := A_S \wedge B_S$*,* $(A \vee B)_S := A_S \vee B_S$ *and* $(\exists x \, A)_S := \exists x \, A_S$*,*

Figure 1: Axioms and rules of SL

Propositional axioms and rules

$$\Gamma \vdash_S \{\alpha \cdot A \cdot \alpha\} \quad \text{if } A^u \in \Gamma \text{ for some } u \qquad \Gamma \vdash_S \{\alpha \cdot \top \cdot \alpha\}$$

$$\frac{\Gamma \vdash_S \{\alpha \cdot A \cdot \beta\} \quad \Gamma \vdash_S \{\beta \cdot B \cdot \gamma\}}{\Gamma \vdash_S \{\alpha \cdot A \wedge B \cdot \gamma\}} \wedge_S I$$

$$\frac{\Gamma \vdash_S \{\alpha \cdot A \wedge B \cdot \beta\}}{\Gamma \vdash_S \{\alpha \cdot A \cdot \beta\}} \wedge_S E_L \qquad \frac{\Gamma \vdash_S \{\alpha \cdot A \wedge B \cdot \beta\}}{\Gamma \vdash_S \{\alpha \cdot B \cdot \beta\}} \wedge_S E_R$$

$$\frac{\Gamma \vdash_S \{\alpha \cdot A \cdot \beta\}}{\Gamma \vdash_S \{\alpha \cdot A \vee B \cdot \beta\}} \vee_S I_L \qquad \frac{\Gamma \vdash_S \{\alpha \cdot B \cdot \beta\}}{\Gamma \vdash_S \{\alpha \cdot A \vee B \cdot \beta\}} \vee_S I_R$$

$$\frac{\Gamma \vdash_S \{\alpha \cdot A \vee B \cdot \beta\} \quad \Gamma, A^u \vdash_S \{\beta \cdot C \cdot \gamma\} \quad \Gamma, B^v \vdash_S \{\beta \cdot C \cdot \gamma\}}{\Gamma \vdash_S \{\alpha \cdot C \cdot \gamma\}} \vee_S E$$

$$\frac{\Gamma, A^u \vdash_S \{\alpha \cdot B \cdot \beta\}}{\Gamma \vdash_S \{\gamma \cdot A \Rightarrow \{\alpha \cdot B \cdot \beta\} \cdot \gamma\}} \Rightarrow_S I \qquad \frac{\Gamma \vdash_S \{\alpha \cdot A \Rightarrow \{\gamma \cdot B \cdot \delta\} \cdot \beta\} \quad \Gamma \vdash_S \{\beta \cdot A \cdot \gamma\}}{\Gamma \vdash_S \{\alpha \cdot B \cdot \delta\}} \Rightarrow_S E$$

$$\frac{\Gamma \vdash_S \{\alpha \cdot \bot \cdot \beta\}}{\Gamma \vdash_S \{\alpha \cdot A \cdot \gamma\}} \bot_S E$$

Quantifier rules

$$\frac{\Gamma \vdash_S \{\alpha[y/x] \cdot A[y/x] \cdot \beta[y/x]\}}{\Gamma \vdash_S \{\gamma \cdot \forall x \{\alpha \cdot A \cdot \beta\} \cdot \gamma\}} \forall_S I \qquad \frac{\Gamma \vdash_S \{\alpha \cdot \forall x \{\beta \cdot A \cdot \gamma\} \cdot \beta[t/x]\}}{\Gamma \vdash_S \{\alpha \cdot A[t/x] \cdot \gamma[t/x]\}} \forall_S E$$

$$\frac{\Gamma \vdash_S \{\alpha \cdot A[t/x] \cdot \beta\}}{\Gamma \vdash_S \{\alpha \cdot \exists x A \cdot \beta\}} \exists_S I \qquad \frac{\Gamma \vdash_S \{\alpha \cdot \exists x A \cdot \beta\} \quad \Gamma, A[y/x]^u \vdash_S \{\beta \cdot C \cdot \gamma\}}{\Gamma \vdash_S \{\alpha \cdot C \cdot \gamma\}} \exists_S E$$

for $\forall_S I$, $y \equiv x$ or y not free in A, α, β, and y not free in Γ
for $\exists_S E$, $y \equiv x$ or y not free in A, and y not free in C, α, β, γ or Γ.

Basic Hoare rules

$$\frac{\alpha \vdash_H \beta \quad \Gamma \vdash_S \{\beta \cdot A \cdot \gamma\} \quad \gamma \vdash_H \delta}{\Gamma \vdash_S \{\alpha \cdot A \cdot \delta\}} \mathit{cons}$$

$$\frac{\vdash_H \alpha \vee \beta \quad \Gamma \vdash_S \{\alpha \wedge \gamma \cdot A \cdot \delta\} \quad \Gamma \vdash_S \{\beta \wedge \gamma \cdot A \cdot \delta\}}{\Gamma \vdash_S \{\gamma \cdot A \cdot \delta\}} \mathit{cond}$$

Additional axioms

state axioms Δ_H of the form $\Gamma \vdash_H \alpha$
main axioms Δ_S of the form $\Gamma \vdash_S \{\alpha \cdot A \cdot \beta\}$

- $(A \Rightarrow B)_S := A_S \Rightarrow \{\top \cdot B_S \cdot \top\}$ *and* $(\forall x\, A)_S := \forall x\, \{\top \cdot A_S \cdot \top\}$.

Then whenever $\Gamma \vdash_I A$ *is provable in* PL*, we have that* $\Gamma_S \vdash_S \{\top \cdot A_S \cdot \top\}$ *is provable in* SL*, where* $\Gamma_S := (A_1)_S^{u_1}, \ldots, (A_n)_S^{u_n}$ *for* $\Gamma := A_1^{u_1}, \ldots, A_n^{u_n}$.

2.3 The intuition behind SL

The intended semantic meaning of $\Gamma \vdash_H \alpha$ is that α can be inferred from the assumptions Γ for any fixed state. The intended meaning of $\Gamma \vdash_S \{\alpha \cdot A \cdot \beta\}$, on the other hand, is that from assumptions Γ, if α holds with respect to some input state, then we can infer that A is true and β holds with respect to some output state (this will be made formal in Section 5 via our realizability interpretation formulated in the metatheory S^ω, and the reader is invited to look ahead to Remark 4.4 to see the semantic meaning of $\{\alpha \cdot A \cdot \beta\}$ couched as a standard logical expression). Crucially, in SL the state is *implicit*, and so there are no variables or terms of state type. The state will rather be made explicit in our metatheory S^ω. The main axioms and rules of SL simply describe how this semantic interpretation propagates in a call-by-value manner through the usual axioms and rules of first-order logic. The state itself is brought into play through the Hoare rules along with the additional axioms Δ_H and Δ_S.

The two Hoare rules of SL correspond to the *consequence* and *conditional* rules of traditional Hoare logic. The usual conditional rule falls out as a special case of ours since we assume $\Gamma \vdash_H \alpha \vee \neg\alpha$. Some of the other traditional Hoare rules are derivable: The empty statement axiom corresponds to our own axiom $\Gamma \vdash_S \{\alpha \cdot \top \cdot \alpha\}$ while composition can be viewed as the special case of $\wedge_S I$ for $A = B = \top$. In Section 6.1 we discuss how controlled while loops can also be added to our logic, for instance if we include axioms and rules for reasoning about natural numbers by instantiating PL as Heyting arithmetic. But for now, we illustrate our logic with some very straightforward scenarios.

Example 2.2 (Simple read-write). Consider a very simple state that allows us to perform the following three actions:

1. Store any value from our domain of discourse in an input location.
2. For the current value x in input location, compute some y such that $P(x, y)$ holds (where P is a fixed binary predicate symbol of the logic), and store it in an output location.
3. Retrieve the computed value y from the state.

We could formalise those three actions by including two unary state predicates stored and solved, and adding the following axioms to Δ_S:

$$\Gamma \vdash_S \{\alpha \cdot \top \cdot \mathsf{stored}(x)\} \quad \text{where } \alpha \text{ ranges over all state formulas}$$
$$\Gamma \vdash_S \{\mathsf{stored}(x) \cdot \top \cdot \mathsf{solved}(x)\}$$
$$\Gamma \vdash_S \{\mathsf{solved}(x) \cdot \exists y\, P(x, y) \cdot \top\}$$

We can then, for example, derive the following in $\mathrm{SL} + \Delta_H + \Delta_S$ for $\Delta_H = \emptyset$, where α, β are any state formulas:

$$\vdash_S \{\beta \cdot \forall x\, \{\alpha \cdot \exists y\, P(x, y) \cdot \top\} \cdot \beta\}$$

An example of such a derivation is given in Appendix B. We note that the state formulas and actions are used in the proof, but if we set $\alpha = \beta = \top$ then the components of the theorem are just formulas in ordinary first-order logic.

Example 2.3 (Fixed-length array sorting). Let us now consider our state as an array of length three, and elements in that array as having some order structure. We formalise this in SL by introducing $1, 2, 3$ as special constant symbols, along with two state predicates: a binary predicate $\leq$ for comparing elements at locations x and y, and a nullary predicate sorted that declares that the state is sorted. These can be

characterised by adding the following axiom schemes, but to Δ_H rather than Δ_S as they do not represent state *actions*:

$$\Gamma \vdash_H 1 \leq 2 \wedge 2 \leq 3 \Rightarrow \mathsf{sorted}$$
$$\Gamma \vdash_H s \leq t \vee t \leq s \quad \text{where } s,t \text{ range over all terms}$$

We then allow a single action on our array, namely the swapping of a pair of elements in the list, which is axiomatised by adding to Δ_S all instances of

$$\Gamma \vdash_S \{\alpha[s/x, t/y] \cdot \top \cdot \alpha[t/x, s/y]\}$$

where α ranges over state formulas and s, t over terms. The statement that all arrays of length three can be sorted is formulated as

$$\vdash_S \{\top \cdot \top \cdot \mathsf{sorted}\}$$

and an example of a proof of this in $\mathrm{SL} + \Delta_H + \Delta_S$ is given in Appendix B. In contrast to Example 2.2 above, this is an example of a purely imperative proof that involves no propositional formulas other than $\top$. As we will see in Example 5.6, the proof corresponds to a purely imperative program.

3 The system ST: A simple functional/imperative term calculus

We now define our calculus $\mathrm{ST} + \Lambda_S$ whose terms will represent realizers for proofs in $\mathrm{SL} + \Delta_H + \Delta_S$. This is a standard typed lambda calculus for mixed functional and imperative programs, and is defined to include basic terms together with additional constants in some set Λ_S, where the latter are intuitively there to realize the axioms in Δ_S. Semantics for the terms will be given via a monadic translation into the metalanguage defined in the next section. Types are defined by the grammmar

$$X ::= D \mid C \mid X \times X \mid X + X \mid X \to X$$

while basic terms are defined as

$$e ::= \mathsf{skip} \mid \mathsf{default}_X \mid c \mid f \mid x \mid p_0(e) \mid p_1(e) \mid e \circ e \mid \iota_0(e) \mid \iota_1(e) \mid \mathsf{elim}\, e\, e\, e \mid \lambda x.e \mid e\, e \mid \mathsf{if}\, \alpha\, \mathsf{then}\, e\, \mathsf{else}\, e$$

where f range over all function symbols of SL, c are constants in Λ_S, and α ranges over state formulas of SL. Typing derivations of the form $\Gamma \vdash t : X$ are given below, where Γ is a set of typed variables. Note that the types of constants $c \in \Lambda_S$ are also left unspecified.

$$\Gamma \vdash f : D^n \to D \quad \text{where } f \text{ has arity } n \qquad \Gamma \vdash c : X$$

$$\Gamma \vdash x : X \quad \text{if } x : X \text{ in } \Gamma \qquad \Gamma \vdash \mathsf{skip} : C$$

$$\frac{\Gamma \vdash s : X \quad \Gamma \vdash t : Y}{\Gamma \vdash s \circ t : X \times Y} \qquad \frac{\Gamma \vdash t : X \times Y}{\Gamma \vdash p_0(t) : X} \qquad \frac{\Gamma \vdash t : X \times Y}{\Gamma \vdash p_1(t) : Y}$$

$$\frac{\Gamma \vdash t : X}{\Gamma \vdash \iota_0(t) : X + Y} \qquad \frac{\Gamma \vdash t : Y}{\Gamma \vdash \iota_1(t) : X + Y} \qquad \frac{\Gamma \vdash r : X + Y \quad \Gamma \vdash s : X \to Z \quad \Gamma \vdash t : Y \to Z}{\Gamma \vdash \mathsf{elim}\, r\, s\, t \vdash Z}$$

$$\frac{\Gamma, x : X \vdash t : Y}{\Gamma \vdash \lambda x.t : X \to Y} \qquad \frac{\Gamma \vdash t : X \to Y \quad \Gamma \vdash s : X}{\Gamma \vdash ts : Y} \qquad \Gamma \vdash \mathsf{default}_X : X$$

$$\frac{\Gamma \vdash s : X \quad \Gamma \vdash t : X \quad x : D \in \Gamma \text{ for all free variables of } \alpha}{\Gamma \vdash \mathsf{if}\, \alpha\, \mathsf{then}\, s\, \mathsf{else}\, t : X}$$

The type C should be interpreted as a type of commands that act on the state but don't return any values. It is be helpful to consider a derived operator for sequential composition:

Definition 3.1. If $\Gamma \vdash s : C$ and $\Gamma \vdash t : X$ then $\Gamma \vdash s * t := p_1(s \circ t) : X$.

4 A monadic embedding of SL and ST into a metatheory S^ω

We now give a semantic interpretation of both state formulas of $\mathrm{SL} + \Delta_H + \Delta_S$ and terms in $\mathrm{ST} + \Lambda_S$ into a standard higher-order, many sorted logic $S^\omega + \Lambda_{S^\omega}$.

4.1 The system S^ω

This logic contains typed lambda terms along with equational axioms for reasoning about them, together with the usual axioms and rules of many-sorted predicate logic. Because most aspects of the logic are completely standard, and in any case it is purely a verifying system, we are less detailed in specifying it. Types are defined as follows:

$$X ::= D \,|\, 1 \,|\, \mathrm{Bool} \,|\, S \,|\, X \times X \,|\, X \to X$$

where D represents objects in the domain of SL (just as in ST), Bool a type of booleans, and states are now explicitly represented as objects of type S. Our metatheory is an equational calculus, with an equality symbol $=_X$ for all types. Typed terms include:

- variables $x, y, z, \ldots$ for each type, where we denote state variables by $\pi, \pi_1, \pi_2, \ldots$
- a constant $f : D^n \to D$ for each n-ary function symbol of SL,
- additional, as yet unspecified constant symbols $c : X$ for interpreting objects in Λ_S, along with axioms that characterise them,
- a unit element $() : 1$ along with the axiom $x = ()$,
- boolean constants t and f, with the axiom $x =_{\mathrm{Bool}} \mathsf{t} \vee x =_{\mathrm{Bool}} \mathsf{f}$,
- pairing $\langle s, t\rangle$ and projection $\mathsf{proj}_0(t)$, $\mathsf{proj}_1(t)$ operators, with the usual axioms,
- terms formed by lambda abstraction and application, with the rule $(\lambda x.t)s = t[s/x]$,
- for each type X a case operator $\mathsf{case}\,(b)\,(s)\,(t)$ for b : Bool and $s, t : X$, with axioms $\mathsf{case}\,\mathsf{f}\,x\,y = x$ and $\mathsf{case}\,\mathsf{t}\,x\,y = y$.

We sometimes write x^X instead of $x : X$, and we use abbreviations such as $\langle x, y, z\rangle$ for $\langle x, \langle y, z\rangle\rangle$. Atomic formulas of S^ω include all ordinary predicate symbols $P, Q, R, \ldots$ of SL as atomic formulas, where an n-ary predicate P in SL takes arguments of type D^n in S^ω, along with predicates $p, q, r, \ldots$ for each state predicate symbol of SL, but now, if p is an n-ary state predicate in SL, p takes arguments of type $D^n \times S$ in S^ω. General formulas are built using the usual logical connectives, including quantifiers for all types. The axioms and rules of S^ω include the axioms of rules of predicate logic (now in all finite types), axioms for the terms, along with the usual equality axioms (including full extensionality). Because S^ω acts as a verifying theory, we freely use strong axioms (such as extensionality), without concerning ourselves with the minimal such system that works.

4.2 The embedding $[\cdot]$ on state formulas of SL

The main purpose of our metalanguage is to allow us to reason semantically about SL and ST. To do this, we introduce an embedding of state formulas of SL and terms of ST into S^ω. We use the same notation $[\cdot]$ for both, as there is no danger of ambiguity.

Definition 4.1. For each term t of SL, there is a natural interpretation of t as a term of type D in ST, namely $x \mapsto x : D$ and $f(t_1, \ldots, t_n) \mapsto f(t_1 \circ \cdots \circ t_n) : D$. Similarly, there is a natural interpretation of t into S^ω, this time with $f(t_1, \ldots, t_n) \mapsto f(\langle t_1, \ldots, t_n\rangle)$. We use the same notation for t in each of the three systems, as there is no risk of ambiguity.

Definition 4.2. For each *state* formula α of , we define a formula $[\alpha](\pi)$ of S^ω, whose free variables are the same as those of α (but now typed with type D) with the potential addition of a single state variable π, as follows:

- $[\top](\pi) := \top$ and $[\bot](\pi) := \bot$,
- $[p(t_1, \ldots, t_n)](\pi) := p(t_1, \ldots, t_n, \pi)$,
- $[\alpha \wedge \beta](\pi) := [\alpha](\pi) \wedge [\beta](\pi)$, and similarly for $\alpha \vee \beta$ and $\alpha \Rightarrow \beta$.

The following Lemma is easily proven using induction over propositional derivations.

Lemma 4.3. *If* $\Gamma \vdash_H \alpha$ *in* SL *then* $[\alpha](\pi)$ *is provable in* S^ω *from the assumptions* $[\Gamma](\pi)$*, where* $[\Gamma](\pi) := [\alpha_1](\pi), \ldots, [\alpha_n](\pi)$ *for* $\Gamma := \alpha_1, \ldots, \alpha_n$*. This extends to proofs in* $\mathrm{SL} + \Delta_H$ *provided that the embedding of any axiom in* Δ_H *is provable in* $S^\omega + \Lambda_{S^\omega}$*.*

Remark 4.4. We would now be in a position to make the semantic meaning of main formulas of SL precise, extending $[\cdot]$ to main formulas of SL by setting

$$[\{\alpha \cdot A \cdot \beta\}] := \exists \pi\, [\alpha](\pi) \Rightarrow [A] \wedge \exists \pi\, [\beta](\pi)$$

and defining the remaining connectives in the obvious way. Similarly to Lemma 4.3, we can then prove that if $\Gamma \vdash_S \{\alpha \cdot A \cdot \beta\}$ then $[\{\alpha \cdot A \cdot \beta\}]$ is provable in S^ω from assumptions $[\Gamma]$. However, technically this is not necessary to formulate our realizability interpretation or prove our soundness theorem, so we omit the details.

4.3 The embedding $[\cdot]$ on terms of ST

Our translation on terms is a call-by-value monadic translation using the state monad $S \to X \times S$. We first define a translation on types of ST as follows:

- $[D] := D$, $[C] := 1$ and $[X \times Y] := [X] \times [Y]$,
- $[X + Y] := \mathrm{Bool} \times [X] \times [Y]$
- $[X \to Y] := [X] \to S \to [Y] \times S$

Lemma 4.5. *For any type* X *of* SL*, the type* $[X]$ *is inhabited, in the sense that we can define a canonical closed term* $0_X : [X]$*.*

Proof. Induction on types, letting $0_D := c$ for a constant symbol which is assumed to exist in SL. The only other nonstandard case is $0_{X \to Y}$, which can be defined as $\lambda x, \pi\,.\,\langle 0_Y, \pi \rangle$. □

Finally, before introducing our translation on terms, we need to add characteristic functions to S^ω for all state formulas (analogous to the characteristic functions for quantifier-free formulas in [12]). For any state formula $\alpha[x_1, \ldots, x_n]$ of SL, where $x_1, \ldots, x_n$ are the free variables of α, we introduce constants $\chi_\alpha : D^n \to S \to X \to X$ satisfying the axioms

$$[\alpha][x_1, \ldots, x_n](\pi) \Rightarrow \chi_\alpha \langle x_1, \ldots, x_n \rangle\, \pi\, y\, z = y$$
$$[\neg\alpha][x_1, \ldots, x_n](\pi) \Rightarrow \chi_\alpha \langle x_1, \ldots, x_n \rangle\, \pi\, y\, z = z$$

Definition 4.6. For each term $\Gamma \vdash t : X$ of ST we define a term $[\Gamma] \vdash [t] : S \to [X] \times S$ of S^ω as follows, where $[\cdot]$ is defined on contexts as $[x_1 : X_1, \ldots, x_n : X_n] := x_1 : [X_1], \ldots, x_n : [X_n]$:

- $[x]\pi := \langle x, \pi\rangle$,
- $[\mathsf{skip}]\pi := \langle (), \pi\rangle$,
- $[f]\pi := \langle \lambda x^{D^n}, \pi\,.\,\langle fx, \pi\rangle, \pi\rangle$,
- $[c]\pi$ is appropriately defined for each additional constant in Λ_S,
- $[s \circ t]\pi := \langle a, b, \pi_2\rangle$ where $\langle a, \pi_1\rangle := [s]\pi$ and $\langle b, \pi_2\rangle := [t]\pi_1$,
- $[p_0 t]\pi := \langle a, \pi_1\rangle$ and $[p_1 t]\pi := \langle b, \pi_1\rangle$ where $\langle a, b, \pi_1\rangle := [t]\pi$,
- $[\iota_0 t]\pi := \langle \mathsf{f}, a, 0_Y, \pi_1\rangle$ and $[\iota_1 t] := \langle \mathsf{t}, 0_X, b, \pi_1\rangle$ for $\langle a, \pi_1\rangle := [t]\pi$,
- $[\mathsf{elim}\, r\, s\, t]\pi := \mathsf{case}\, e\, (fa\pi_2)\, (gb\pi_3)$ for $\langle e, a, b, \pi_1\rangle := [r]\pi$, $\langle f, \pi_2\rangle := [s]\pi_1$, $\langle g, \pi_3\rangle := [t]\pi_1$,
- $[\lambda x.t]\pi := \langle \lambda x^{[X]}.[t], \pi\rangle$,
- $[ts]\pi := fa\pi_2$ for $\langle f, \pi_1\rangle := [t]\pi$ and $\langle a, \pi_2\rangle := [s]\pi_1$,
- $[\mathsf{default}_X]\pi := \langle 0_X, \pi\rangle$,
- $[\mathsf{if}\, \alpha[x_1, \ldots, x_n]\, \mathsf{then}\, s\, \mathsf{else}\, t]\pi := \chi_\alpha\, \langle x_1, \ldots, x_n\rangle\, \pi\, ([s]\pi)\, ([t]\pi)$ where $\{x_1, \ldots, x_n\}$ are the free variables of α.

The following lemmas will be useful when verifying our realizability interpretation in the next section. The first is by a simple induction on terms.

Lemma 4.7. *For any term t of* SL*, we have $[t]\pi = \langle t, \pi\rangle$ (cf. Definitions 4.1 and 4.6).*

Lemma 4.8 (Currying in ST)**.** *Suppose that $\Gamma, x : X, y : Y \vdash t : Z$ is a term in* ST*, and define $\Gamma \vdash \lambda^* v.t : X \times Y \to Z$ by $\lambda^* v.t := \lambda v.(\lambda x, y.t)(p_0 v)(p_1 v)$ where v is not free in t. Then for any $s : X \times Y$ we have*

$$[(\lambda^* v.t)s]\pi = [t][a/x, b/y]\pi_1$$

where $\langle a, b, \pi_1\rangle := [s]\pi$.

Proof. By unwinding the definition of $[\cdot]$. Full details are given in Appendix D. □

5 A realizability interpretation of SL into ST

We now come to the main contribution of the paper, which is the definition of a realizability relation between terms of ST and formulas of SL, along with a soundness theorem that shows us how to extract realizers from proofs. Our metatheory S^ω is used to define the realizability relation and prove the soundness theorem.

Definition 5.1 (Types of realizers). To each main formula A of SL we assign a type $\tau_S(A)$ of ST as follows:

- $\tau_S(\top) = \tau_S(\bot) = \tau_S(P(t_1, \ldots, t_n)) := C$,
- $\tau_S(A \wedge B) := \tau_S(A) \times \tau_S(B)$,
- $\tau_S(A \vee B) := \tau_S(A) + \tau_S(B)$,
- $\tau_S(\exists x\, A) := D \times \tau_S(A)$,
- $\tau_S(A \Rightarrow \{\alpha \cdot B \cdot \beta\}) := \tau_S(A) \to \tau_S(B)$,
- $\tau_S(\forall x\, \{\alpha \cdot A \cdot \beta\}) := D \to \tau_S(A)$.

Definition 5.2 (Realizability relation). For each main formula A of SL we define a formula $x \text{ sr } A$ of S^ω, whose free variables are contained in those of A (now typed with type D) together with a fresh variable $x : [\tau_S(A)]$, by induction on the structure of A as follows:

- $x \text{ sr } Q := Q$ for $Q = \top, \bot$ or $P(t_1, \ldots, t_n)$,
- $x \text{ sr } A \wedge B := (\mathsf{proj}_0 x \text{ sr } A) \wedge (\mathsf{proj}_1 x \text{ sr } B)$,
- $x \text{ sr } A \vee B := (\mathsf{proj}_0 x = \mathsf{f} \Rightarrow \mathsf{proj}_0(\mathsf{proj}_1 x) \text{ sr } A) \wedge (\mathsf{proj}_0 x = \mathsf{t} \Rightarrow \mathsf{proj}_1(\mathsf{proj}_1 x) \text{ sr } B)$,
- $x \text{ sr } \exists y\, A(y) := (\mathsf{proj}_1 x \text{ sr } A)[\mathsf{proj}_0 x / y]$,
- $f \text{ sr } (A \Rightarrow \{\alpha \cdot B \cdot \beta\}) := \forall x^{[\tau_S(A)]}\, (x \text{ sr } A \Rightarrow fx \text{ sr } \{\alpha \cdot B \cdot \beta\})$,
- $f \text{ sr } (\forall x\, \{\alpha(x) \cdot A(x) \cdot \beta(x)\}) := \forall x^D\, (fx \text{ sr } \{\alpha(x) \cdot A(x) \cdot \beta(x)\})$,

where for $x : S \to [\tau_S(A)] \times S$ we define

- $x \text{ sr } \{\alpha \cdot A \cdot \beta\} := \forall \pi^S\, ([\alpha](\pi) \Rightarrow \mathsf{proj}_0(x\pi) \text{ sr } A \wedge [\beta](\mathsf{proj}_1(x\pi)))$.

The following substitution lemma is easily proven by induction on formulas of SL.

Lemma 5.3. *For any term t of* SL *and $s : [\tau_S(A)]$ we have $s \text{ sr } A[t/x] = (s \text{ sr } A)[t/x]$, where x is not free in s and on the right hand side we implicitly mean the natural interpretation of t in S^ω (cf. Definition 4.1).*

Theorem 5.4 (Soundness)**.** *Suppose that*

$$\Gamma := A_1^{u_1}, \ldots, A_n^{u_n} \vdash_S \{\alpha \cdot A \cdot \beta\}$$

is provable in SL*. Then we can extract from the proof a term $\Delta, \tau_S(\Gamma) \vdash t : \tau_S(A)$, where Δ contains the free variables of Γ and $\{\alpha \cdot A \cdot \beta\}$ (typed with type D) and $\tau_S(\Gamma) := x_1 : \tau_S(A_1), \ldots, x_n : \tau_S(A_n)$, such that*

$$[t] \text{ sr } \{\alpha \cdot A \cdot \beta\}$$

is provable in S^ω from the assumptions $(x_1 \text{ sr } A_1)^{u_1}, \ldots, (x_n \text{ sr } A_n)^{u_n}$ for $x_i : [\tau_S(A_i)]$. The theorem holds more generally for proofs in $\mathrm{SL} + \Delta_H + \Delta_S$*, now provably in $S^\omega + \Lambda_{S^\omega}$, if:*

- *for any axiom $\Gamma \vdash_H \alpha$ in Δ_H, the corresponding axiom $[\Gamma](\pi) \Rightarrow [\alpha](\pi)$ is added to Λ_{S^ω},*
- *for any axiom in Δ_S there is a term t of $\mathrm{ST} + \Lambda_S$ such that $[t]$ realizes that axiom provably in $S^\omega + \Lambda_{S^\omega}$.*

Proof. Induction on the structure of derivations in SL. Most of the axioms and rules are straightforward, and those are covered in Appendix D. In all cases, we assume as global assumptions $(x_1 \text{ sr } A_1)^{u_1}, \ldots, (x_n \text{ sr } A_n)^{u_n}$, and our aim is then to produce a term t such that if $[\alpha](\pi)$ holds for some state variable π, then $a \text{ sr } A$ and $[\beta](\pi_1)$ hold for $\langle a, \pi_1 \rangle := [t]\pi$.

- $(\vee_S E)$ Suppose that r, $s(x)$ and $t(y)$ are such that $[r] \text{ sr } \{\alpha \cdot B \cdot \beta\}$, $[s](x) \text{ sr } \{\beta \cdot C \cdot \gamma\}$ assuming $x \text{ sr } A$, and $[t](y) \text{ sr } \{\beta \cdot C \cdot \gamma\}$ assuming $y \text{ sr } B$. We claim that

  $$[\mathsf{elim}\, r\, (\lambda x.s)\, (\lambda y.t)] \text{ sr } \{\alpha \cdot C \cdot \gamma\}$$

  To prove this, first note that if $[\alpha](\pi)$, we have $\langle e, a, b \rangle \text{ sr } A \vee B$ and $[\beta](\pi_1)$ for $\langle e, a, b, \pi_1 \rangle := [r]\pi$. There are now two possibilities. If $e = \mathsf{f}$ then

  $$\begin{aligned} [\mathsf{elim}\, r\, (\lambda x.s)\, (\lambda y.t)]\pi = fa\pi_2 \quad \text{for } \langle f, \pi_2 \rangle := [\lambda x.s]\pi_1 = \langle \lambda x.[s](x), \pi_1 \rangle \\ = (\lambda x.[s](x))a\pi_1 \\ = [s](a)\pi_1 \end{aligned}$$

But since $[\beta](\pi_1)$ holds and $e = \mathsf{f}$ also implies that a sr A, we have c sr C and $[\gamma](\pi_2)$ for $\langle c, \pi_2\rangle := [s](a)\pi_1$, which proves the main claim in the case $e = \mathsf{f}$. An analogous argument works for the case $e = \mathsf{t}$.

- $(\exists_S E)$ Suppose that s and $t(x, z)$ are such that $[s]$ sr $\{\alpha \cdot \exists x\, A \cdot \beta\}$ and

$$z \text{ sr } A[y/x] \Rightarrow [t](y, z) \text{ sr } \{\beta \cdot C \cdot \gamma\}$$

where $y \equiv x$ or y is not free in A, and y is also not free in C, α, β, γ or Γ. By Lemma 5.3 that z sr $A[y/x] = (z \text{ sr } A)[y/x] = \langle y, z\rangle$ sr $\exists x\, A$ we therefore have

$$\langle y, z\rangle \text{ sr } \exists x\, A \Rightarrow [t](y, z) \text{ sr } \{\beta \cdot C \cdot \gamma\}$$

Now, applying Lemma 4.8 to $\Delta, \Gamma, y : D, z : \tau_S(A) \vdash t : \tau_S(C)$, we have

$$[(\lambda^* v.t)s]\pi = [t](e, a)\pi_1$$

for $\langle e, a, \pi_1\rangle := [s]\pi$. Now, if $[\alpha](\pi)$ holds, then we have $\langle e, a\rangle$ sr $\exists x\, A$ and $[\beta](\pi_1)$, and therefore since $[t](e, a)$ sr $\{\beta \cdot C \cdot \gamma\}$, we have c sr C and $[\gamma](\pi_2)$ for $\langle c, \pi_2\rangle = [t](e, a)\pi_1 = [(\lambda^* v.t)s]\pi$, and thus we have shown that $[(\lambda^* v.t)s]$ sr $\{\alpha \cdot C \cdot \gamma\}$.

- (*cond*) Suppose that $[s]$ sr $\{\alpha \wedge \gamma \cdot A \cdot \delta\}$ and $[t]$ sr $\{\beta \wedge \gamma \cdot A \cdot \delta\}$. We claim that

$$[\mathsf{if}\ \alpha\ \mathsf{then}\ s\ \mathsf{else}\ t] \text{ sr } \{\gamma \cdot A \cdot \delta\}$$

To prove this, suppose that $[\gamma](\pi)$ holds. Since $\vdash_H \alpha \vee \beta$ then $[\alpha](\pi) \vee [\beta](\pi)$ is provable in S^ω, and so we consider two cases. Let $\{x_1, \ldots, x_n\}$ be the free variables of α. If $[\alpha](\pi)$ holds, then

$$[\mathsf{if}\ \alpha\ \mathsf{then}\ s\ \mathsf{else}\ t]\pi = \chi_\alpha \langle x_1, \ldots, x_n\rangle\, \pi\, ([s]\pi)\, ([t]\pi) = [s]\pi$$

and since then $[\alpha](\pi) \wedge [\gamma](\pi)$ we have a sr A and $[\delta](\pi_1)$ for $\langle a, \pi_1\rangle := [s]\pi$. On the other hand, if $[\beta](\pi)$ holds, then by an analogous argument we can show that a sr A and $[\delta](\pi_1)$ for $\langle a, \pi_1\rangle := [t]\pi = [\mathsf{if}\ \alpha\ \mathsf{then}\ s\ \mathsf{else}\ t]\pi$, and we are done.

The extension of the soundness theorem to $\mathrm{SL} + \Delta_H + \Delta_S$ is straightforward, as the soundness proof is modular and so any axioms along with their realizers can be added. The first condition is needed so that Lemma 4.3 (needed for the *cons* rule) continues to apply. □

Corollary 5.5 (Program extraction)**.** *Suppose that the sentence*

$$\vdash_S \{\alpha \cdot \forall x\, \{\beta \cdot \exists y\, P(x, y) \cdot \gamma(x)\} \cdot \beta\}$$

is provable in $\mathrm{SL} + \Delta_S$*. Then we can extract a closed realizing term* $t : D \to D \times C$ *in* $\mathrm{ST} + \Lambda_S$ *such that defining* $g : D \to S \to D \times S$ *by* $gx\pi := \langle a, \pi_2\rangle$ *for* $\langle f, \pi_1\rangle := [t]\pi$ *and* $\langle a, (), \pi_2\rangle := fx\pi_1$*, we have*

$$\forall \pi^S([\alpha](\pi) \Rightarrow \forall x^D\, (P(x, \mathsf{proj}_0(gx\pi)) \wedge [\gamma](x)(\mathsf{proj}_1(gx\pi))))$$

provably in $S^\omega + \Lambda_{S^\omega}$*.*

5.1 Simplification and removal of unit types

In presentations of modified realizability that use product types instead of type sequences, it is common to introduce the notion of a Harrop formula (a formula that does not contain disjunction or existential quantification in a positive position) and define realizability in a way that all Harrop formulas have unit realizability type, so that e.g. $\tau_S(\forall x\,(P \wedge Q)) = 1$ for atomic predicates P and Q, rather than $\tau_S(\forall x\,(P \wedge Q)) = D \to 1{\times}1$ as for us. We have avoided this simplification earlier on, as it would have added additional cases and bureaucracy to our soundness theorem. However, we can compensate retroactively for this choice by introducing equivalences on types that eliminate unit types, namely the closure under contexts of

$$1 \times X \simeq 1 \simeq X \times 1 \quad (1 \to X) \simeq X \quad (X \to 1) \simeq 1$$

along with corresponding equivalences on terms, also closed under contexts:

$$t^{1\times X} \simeq \mathsf{proj}_1(t)^X \quad t^{X\times 1} \simeq \mathsf{proj}_0(t)^X \quad t^{1\to X} \simeq t() \quad t^X \simeq \lambda x^1.t \quad t^{X\to 1} \simeq ()$$

For example, in Corollary 5.5 we would then have

$$[t]\pi : (D \to S \to D \times 1 \times S) \times S \simeq (D \to S \to D \times S) \times S \quad \text{and} \quad gx\pi \simeq fx\pi_1.$$

5.2 Examples of program extraction

We now continue the short illustrative examples we outlined in Section 2.3.

Example 5.6 (Simple read-write). In Example 2.2 we considered a state where three actions were possible (writing to the state, performing a calculation, and reading the output from the state). We can formalise these three actions semantically in the metatheory S^ω by including three constants in Λ_{S^ω}, namely $c_1 : D \to S \to S$, $c_2 : S \to S$ and $c_3 : S \to D$, along with the characterising axioms:

1. $\mathsf{stored}(x, c_1 x\pi)$,
2. $\mathsf{stored}(x, \pi) \Rightarrow \mathsf{solved}(x, c_2\pi)$,
3. $\mathsf{solved}(x, \pi) \Rightarrow P(x, c_3\pi)$.

While we are able to use these constants to form non-sequential terms in S^ω, such as $\lambda\pi, \pi_1, x\,.\,\langle c_1 x\pi, c_2\pi_1\rangle$, we can force them to be applied in a sequential manner by adding three corresponding constants to our term calculus ST, namely including $\mathsf{write} : D \to C$, $\mathsf{calc} : C$ and $\mathsf{read} : D \times C$ in Λ_S, along with the embedding rules

- $[\mathsf{write}]\pi := \langle \lambda x, \pi\,.\,\langle (), c_1 x\pi\rangle, \pi\rangle \simeq \langle c_1, \pi\rangle$,
- $[\mathsf{calc}]\pi := \langle (), c_2\pi\rangle$ so that $[\mathsf{calc}] \simeq c_2$,
- $[\mathsf{read}]\pi := \langle c_3\pi, (), \pi\rangle \simeq \langle c_3\pi, \pi\rangle$.

We can then prove the following in S^ω i.e. that all axioms in Δ_S can be realised:

- $[\mathsf{write}(x)]$ sr $\{\alpha \cdot \top \cdot \mathsf{stored}(x)\}$,
- $[\mathsf{calc}]$ sr $\{\mathsf{stored}(x) \cdot \top \cdot \mathsf{solved}(x)\}$,
- $[\mathsf{read}]$ sr $\{\mathsf{solved}(x) \cdot \exists y\, P(x, y) \cdot \top\}$.

and thus Theorem 5.4 applies to $\mathrm{SL} + \Delta_H + \Delta_S$ for $\Delta_H = \emptyset$. In particular, we have

$$[t] \text{ sr } \{\beta \cdot \forall x \, \{\alpha \cdot \exists y \, P(x, y) \cdot \top\} \cdot \beta\}$$

for $t := \lambda x \, . \, ((\mathsf{write}(x) * \mathsf{calc}) * \mathsf{read})$ where $*$ is sequential composition operator from Definition 3.1. A formal derivation of this term from the corresponding proof can found in Appendix C.1.

Example 5.7 (Fixed-length array sorting). In Example 2.3 we considered a situation where we are allowed a single action on our state, namely to swap elements. Analogously to the previous example, we can formalise this in our semantic environment S^ω by adding to Λ_{S^ω} a constant $c : D \times D \to S \to S$ along with the axiom

$$[\alpha[s/x, t/y]](\pi) \Rightarrow [\alpha[t/x, s/y]](c\langle s, t\rangle\pi)$$

ranging over terms s, t of SL, together with axioms corresponding to those of Δ_H i.e.

$$[1 \leq 2 \wedge 2 \leq 3](\pi) \Rightarrow \mathsf{sorted}(\pi) \quad \text{and} \quad [s \leq t \vee t \leq s](\pi)$$

where α ranges over state formulas and s, t over terms of SL. Similarly, we add a term $\mathsf{swap} : D \times D \to C$ to Λ_S and define $[\mathsf{swap}]\pi := \langle \lambda x, \pi \, . \, \langle (), cx\pi\rangle, \pi\rangle \simeq \langle c, \pi\rangle$ so that

$$[\mathsf{swap}(s \circ t)] \text{ sr } \{\alpha[s/x, t/y] \cdot \top \cdot \alpha[t/x, s/y]\}$$

for any terms s, t of SL. A derivation of a closed term $t : C$ of $\mathrm{ST} + \{\mathsf{swap}\}$ such that $[t]$ sr $\{\top \cdot \top \cdot \mathsf{sorted}\}$ is given in Appendix C.2. In particular, we can prove in S^ω that $\forall \pi^S \, \mathsf{sorted}(\mathsf{proj}_1([t]\pi))$, and so the term $\lambda\pi \, . \, \mathsf{proj}_1([t]\pi) : S \to S$ acts as a sorting program for arrays of length three.

6 Directions for future work

In this paper we have presented the central ideas behind a new method for extracting stateful programs from proofs, which include an extension of ordinary first-order logic with Hoare triples, a corresponding realizability interpretation, and a soundness theorem. We emphasise once again that our intention has been to offer an alternative approach to connecting proofs with stateful programs, one that seeks to complement rather than improve existing work by embracing simplicity and abstraction, and which might be well suited to a range of applications in proof theory or computability theory (though we certainly do not want to exclude verification as a potential direction for future work). In this spirit, we conclude with a slightly more detailed description of ways in which this paper could be extended, but in each case leave the details to future work!

6.1 Arithmetic, recursion and while-loops

While our main results have been presented in the neutral setting of first-order logic, it would be straightforward to extend SL to a logic capable of reasoning about and carrying out recursion on more complex datatypes. An obvious next step would be to instantiate SL as a stateful version of Heyting arithmetic, with zero and successor constants, a binary equality predicate along with a lifting of the usual equality axioms to the stateful setting, and finally the following rule for 'stateful induction':

$$\frac{\Gamma \vdash_S \{\alpha \cdot A(0) \cdot \beta(0)\} \quad \Gamma, A(x) \vdash_S \{\beta(x) \cdot A(x+1) \cdot \beta(x+1)\}}{\Gamma \vdash_S \{\gamma \cdot \forall x \, \{\alpha \cdot A(x) \cdot \beta(x)\} \cdot \gamma\}} \, ind$$

This would be realized by adding a standard recursor to ST, which would be interpreted in S^ω in the obvious way, provided we also extend the latter to a typed theory of arithmetic with induction and recursion on natural numbers. Induction would give us access to standard recursive functions in our mixed functional/imperative language, including recursive operations on the state. However, a more interesting and

important addition to our logic and programming language would be the ability to construct and reason about total while-loops. We could add to SL controlled variants of Hoare logic's while rule, such as:

$$\frac{\mathcal{A}_1 \quad \mathcal{A}_2 \quad \mathcal{A}_3}{\Gamma, A(0) \vdash_S \{\alpha(0) \cdot B \cdot \beta\}}\ while$$

where the three hypotheses are:

- $\mathcal{A}_1 := \Gamma, A(x), x < N \vdash_S \{\gamma \wedge \alpha(x) \cdot A(x+1) \cdot \alpha(x+1)\}$,
- $\mathcal{A}_2 := \Gamma, A(x), x < N \vdash_S \{\neg\gamma \wedge \alpha(x) \cdot B \cdot \beta\}$,
- $\mathcal{A}_3 := \Gamma, A(x), x \geq N \vdash_S \{\alpha(x) \cdot B \cdot \beta\}$

This would then be realized by a controlled while-loop in ST, which would be semantically interpreted in the natural way in S^ω. Already, the addition of standard recursion and loops over natural numbers would allow us to extend Examples 2.3 and 5.7 to extract a range of in-place sorting algorithms using our abstract axiomatisation of an ordered state, in a similar spirit to [5]. However, further extensions are naturally possible, including the addition of general fixpoint operators and non-controlled while loops, which would then require a S^ω to be replaced by a domain theoretic semantics that allows for partiality. By implementing all of this in a proof assistant, we would have at our disposal a new technique for synthesising correct-by-construction imperative programs. While this is unlikely to compete with existing techniques for verifying imperative programs, it could be well suited to synthesising and reasoning about programs in specific domains, bearing in mind that our logic gives us considerable flexibility as to how the state is implemented.

6.2 Bar recursion and the semantics of extracted programs

Two of the main starting points for this paper, the monadic realizability of Birolo [6] and the author's own Dialectica interpretation with state [22], address the broader problem of trying to understand the operational semantics of programs extracted from proofs as stateful procedures (the origins and development of this general idea, from Hilbert's epsilon calculus onwards, is brilliantly elucidated in Chapter 1 of Aschieri's thesis [2], who then sets out his own realizability interpretation based on learning). A number of case studies by the author and others [19, 18, 23, 24] have demonstrated that while terms extracted from nontrivial proofs can be extremely complex, they are often much easier to understand if one focuses on the way they interact with the mathematical environment. For example, in understanding a program extracted from a proof using Ramsey's theorem for pairs [18], it could be illuminating to study the *trace* of the program as it queries a colouring at particular pairs, as this can lead to a simpler characterisation of the *algorithm* ultimately being implemented by the term.

While the aforementioned analysis of programs has always been done in an informal way, our stateful realizability interpretation would in theory allow us to extract programs which store this trace formally in the state, where our abstract characterisation of state would allow us to implement it in whichever way is helpful in a given setting. For example, in the case of the Bolzano-Weierstrass theorem [19], our state might record information of the form $x_n \in I$, collecting information about the location of sequence elements. For applications in algebra [24], one might instead store information about a particular maximal ideal.

The aforementioned theorems are typically proven using some form of choice or comprehension, and that in itself leads to the interesting prospect of introducing both stateful recursors and while-loops that are computationally equivalent to variants of *bar recursion* [27]. In [21], several bar recursive programs that arise from giving a computational interpretation to arithmetical comprehension principles are formulated as simple while loops, and these could in principle be incorporated into our system with new controlled Hoare rules in the style of update recursion [4], that replace the conditions $n < N$ and $n \geq N$ in the $\mathcal{A}_i$ above with e.g. $n \in \mathrm{dom}(f)$ and $n \notin \mathrm{dom}(f)$, where f is some partial approximation to a comprehension function. An exploration of such while-loops from the perspective of higher-order computability theory might well be of interest in its own right.

6.3 A logic for probabilistic lambda calculi

Probabilistic functional languages are a major topic of research at present. While work in this direction dates back to the late 1970s [14, 26] where it typically had a semantic flavour, a more recent theme [8, 9, 10] has been to study simple extensions of the lambda calculus with nondeterministic choice operators $\oplus$, where $s \oplus t$ evaluates nondeterministically (or probabilistically) to either s or t. While such calculi have been extensively studied, corresponding *logics* that map under some proof interpretation to probabilistic programs are far more rare (although there is some recent work in this direction e.g. [1]).

We can offer a bridge between logic and probabilistic computation by incorporating probabilistic disjunctions into our logic SL and taking states to be streams of probabilities together with a current 'counter' that increases each time a probabilistic event occurs. In a simple setting where only two outcomes are possible, we can axiomatise this within SL by adding zero and successor functions (allowing us to create numerals n), along with a unary state predicate $\mathsf{count}(n)$. We can then model probabilistic events by adding the appropriate axioms to Δ_S. Suppose, for example, we add two predicate constants $H(x)$ and $T(x)$ (for *heads* and *tails*), along with constants $c_1, c_2, \ldots$ representing coins. Then flipping a coin would be represented by the axiom schema

$$\Gamma \vdash_S \{\mathsf{count}(n) \cdot H(c_i) \vee T(c_i) \cdot \mathsf{count}(n+1)\}$$

where n ranges over numerals and c_i over coin constants, the counter indicating that a probabilistic event has occurred. The act of reading a probability from the state could be interpreted semantically by introducing a constant $\omega : S \to \mathrm{Bool} \times S$ to S^ω, with the axiom

$$\mathsf{count}(n, \pi) \Rightarrow (e = \mathsf{f} \Rightarrow H(c_i)) \wedge (e = \mathsf{t} \Rightarrow T(c_i)) \wedge \mathsf{count}(n+1, \pi_1) \text{ for } \langle e, \pi_1 \rangle := \omega\pi$$

(alternatively, we could simply define $S := \mathrm{Nat} \times (\mathrm{Nat} \to \mathrm{Bool})$ for a type of Nat natural numbers, and also define $\omega\langle n, a\rangle := \langle a(n), \langle n+1, a\rangle\rangle$ and $\mathsf{count}(n, \langle m, a\rangle) := m =_{\mathrm{Nat}} n$).

A probabilistic choice operator $\oplus$ can then be added to the language of ST, along with the typing rule $\Gamma \vdash s \oplus t : X + Y$ for $\Gamma \vdash s : X$ and $\Gamma \vdash t : Y$, and the interpretation

$$[s \oplus t]\pi := \mathsf{case}\, e\, ([\iota_0 s]\pi_1)\, ([\iota_1 t]\pi_1) \text{ where } \langle e, \pi_1 \rangle := \omega\pi$$

In particular, defining $\mathsf{flip} := \mathsf{skip} \oplus \mathsf{skip} : C + C$ we would have

$$[\mathsf{flip}]\ \mathrm{sr}\ \{\mathsf{count}(n) \cdot H(c_i) \vee T(c_i) \cdot \mathsf{count}(n+1)\}$$

although we stress that the operator $\oplus$ and would allow for much more complex probabilistic disjunctions, potentially involving additional computational content.

We are then able to extract probabilistic programs from proofs. For instance, including a winner predicate $W(x)$, two player constant symbols p_1, p_2, and adding axioms $H(c_1), H(c_2) \vdash_S \{\alpha \cdot W(p_1) \cdot \alpha\}$; $T(c_1), T(c_2) \vdash_S \{\alpha \cdot W(p_1) \cdot \alpha\}$; $H(c_1), T(c_2) \vdash_S \{\alpha \cdot W(p_2) \cdot \alpha\}$ and $T(c_1), H(c_2) \vdash_S \{\alpha \cdot W(p_2) \cdot \alpha\}$ for any α, we could prove

$$\vdash_S \{\mathsf{count}(n) \cdot \exists x\, W(x) \cdot \mathsf{count}(n+2)\}$$

expressing the fact that a winner can be determined after two flips. We can then extract a corresponding probabilistic term for realizing this statement.

References

[1] Antonelli, M., Dal Lago, U., and Pistone, P. Curry and Howard meet Borel. Preprint, 2022.

[2] Aschieri, F. *Learning, Realizability and Games in Classical Arithmetic.* PhD thesis, Università degli Studi di Torino and Queen Mary, University of London, 2011.

[3] Berardi, S., and de'Liguoro, U. Toward the interpretation of non-constructive reasoning as non-monotonic learning. *Information and Computation 207*, 1 (2009), 63–81.

[4] Berger, U. A computational interpretation of open induction. In *Proceedings of Logic in Computer Science (LICS '04)* (2004), IEEE, pp. 326–334.

[5] Berger, U., Seisenberger, M., and Woods, G. J. M. Extracting imperative programs from proofs: In-place quicksort. In *Proceedings of Types for Proofs and Programs (TYPES'13)* (2014), vol. 26 of *LIPIcs*, pp. 84–106.

[6] Birolo, G. *Interactive Realizability, Monads and Witness Extraction.* PhD thesis, Università degli Studi di Torino, 2012.

[7] Chlipala, A., Malecha, G., Morrisett, G., Shinnar, A., and Wisnesky, R. Effective interactive proofs for higher-order imperative programs. In *Proceedings of International Conference on Functional Programming (ICFP'09)* (2009), ACM, pp. 79–90.

[8] Dal Lago, U., and Zorzi, M. Probabilistic operational semantics for the lambda calculus. *RAIRO–Theoretical Informatics and Applications 46*, 3 (2012), 413–450.

[9] Deliguoro, U., and Piperno, A. Nondeterministic extensions of untyped λ-calculus. *Information and Computation 122*, 2 (1995), 149–177.

[10] Di Pierro, A., Hankin, C., and Wiklicky, H. Probabilistic lambda-calculus and quantitative program analysis. *Journal of Logic and Computation 15*, 2 (2005), 159–179.

[11] Filliâtre, J.-C. Verification of non-functional programs using interpretations in type theory. *Journal of Functional Programming 13*, 4 (2003), 709–745.

[12] Gerhardy, P., and Kohlenbach, U. Extracting Herbrand disjunctions by functional interpretation. *Archive for Mathematical Logic 44* (2005), 633–644.

[13] Hoare, C. A. R. An axiomatic basis for computer programming. *Communications of the ACM 12*, 10 (1969), 576–580.

[14] Jones, C., and Plotkin, G. A probabilistic powerdomain of evaluations. In *Proceedings of Logic in Computer Science (LICS'89)* (1989), IEEE Press, pp. 186–195.

[15] Kreisel, G. Interpretation of analysis by means of functionals of finite type. In *Constructivity in Mathematics*, A. Heyting, Ed. North-Holland, Amsterdam, 1959, pp. 101–128.

[16] Krivine, J.-L. Realizability in classical logic in interactive models of computation and program behaviour. *Panoramas et synthéses 27* (2009).

[17] Nanevski, A., Ahmed, A., Morrisett, G., and Birkedal, L. Abstract predicates and mutable ADTs in Hoare type theory. In *Proceedings of the European Symposium on Programming (ESOP'07)* (2007), vol. 4421 of *LNCS*, pp. 189–204.

[18] Oliva, P., and Powell, T. A constructive interpretation of Ramsey's theorem via the product of selection functions. *Mathematical Structure in Computer Science 25*, 8 (2015), 1755–1778.

[19] Oliva, P., and Powell, T. A game-theoretic computational interpretation of proofs in classical analysis. In *Gentzen's Centenary: The Quest for Consistency*, R. Kahle and M. Rathjen, Eds. Springer, 2015, pp. 501–531.

[20] Poernomo, I., Crossley, J. N., and Wirsing, M. *Adapting Proofs-as-Programs.* Monographs in Computer Science. Springer, 2005.

[21] Powell, T. Gödel's functional interpretation and the concept of learning. In *Proceedings of Logic in Computer Science (LICS '16)* (2016), ACM, pp. 136–145.

[22] Powell, T. A functional interpretation with state. In *Proceedings of Logic in Computer Science (LICS '18)* (2018), ACM, pp. 839–848.

[23] Powell, T. Well quasi-orders and the functional interpretation. In *Well Quasi-Orders in Computation, Logic, Language and Reasoning*, P. Schuster, M. Seisenberger, and A. Weiermann, Eds., vol. 53 of *Trends in Logic*. Springer, 2020, pp. 221–269.

[24] Powell, T., Schuster, P., and Wiesnet, F. A universal algorithm for krull's theorem. *Information and Computation 287* (2022), 104761.

[25] Reynolds, J. C. Separation logic: a logic for shared mutable data structures. In *Proceedings of Logic in Computer Science (LICS'02)* (2002), IEEE, pp. 55–74.

[26] Saheb-Djaromi, N. Probabilistic LCF. In *Proceedings of International Symposium on Mathematical Foundations of Computer Science (MFCS'78)* (1978), vol. 64 of *LNCS*, pp. 442–451.

[27] Spector, C. Provably recursive functionals of analysis: a consistency proof of analysis by an extension of principles in current intuitionistic mathematics. In *Recursive Function Theory: Proc. Symposia in Pure Mathematics* (1962), F. D. E. Dekker, Ed., vol. 5, American Mathematical Society, pp. 1–27.

[28] Yoshida, N., Honda, K., and Berger, M. Logical reasoning for higher-order functions with local state. In *Proceedings of Foundations of Software Science and Computation Structures (FOSSACS'07)* (2007), vol. 44213 of *LNCS*, pp. 361–377.

A Axioms and rules of PL

Propositional logic

$$\Gamma \vdash_I A \quad \text{if } A^u \in \Gamma \text{ for some } u \qquad \Gamma \vdash_I \top$$

$$\frac{\Gamma \vdash_I A \quad \Gamma \vdash_I B}{\Gamma \vdash_I A \wedge B}\,{\scriptstyle \wedge I} \qquad \frac{\Gamma \vdash_I A \wedge B}{\Gamma \vdash_I A}\,{\scriptstyle \wedge E_L} \qquad \frac{\Gamma \vdash_I A \wedge B}{\Gamma \vdash_I B}\,{\scriptstyle \wedge E_R}$$

$$\frac{\Gamma \vdash_I A}{\Gamma \vdash_I A \vee B}\,{\scriptstyle \vee I_L} \qquad \frac{\Gamma \vdash_I B}{\Gamma \vdash_I A \vee B}\,{\scriptstyle \vee I_R} \qquad \frac{\Gamma \vdash_I A \vee B \quad \Gamma, A^u \vdash_I C \quad \Gamma, B^v \vdash_I C}{\Gamma \vdash_I C}\,{\scriptstyle \vee E}$$

$$\frac{\Gamma, A^u \vdash_I B}{\Gamma \vdash_I A \Rightarrow B}\,{\scriptstyle \Rightarrow I} \qquad \frac{\Gamma \vdash_I A \Rightarrow B \quad \Gamma \vdash_I A}{\Gamma \vdash_I B}\,{\scriptstyle \Rightarrow E} \qquad \frac{\Gamma \vdash_I \bot}{\Gamma \vdash_I A}\,{\scriptstyle \bot E}$$

Quantifier rules

$$\frac{\Gamma \vdash_I A[y/x]}{\Gamma \vdash_I \forall x A}\,{\scriptstyle \forall I} \qquad \frac{\Gamma \vdash_I \forall x A}{\Gamma \vdash_I A[t/x]}\,{\scriptstyle \forall E}$$

$$\frac{\Gamma \vdash_I A[t/x]}{\Gamma \vdash_I \exists x A}\,{\scriptstyle \exists I} \qquad \frac{\Gamma \vdash_I \exists x A \quad \Gamma, A[y/x]^u \vdash_I C}{\Gamma \vdash_I C}\,{\scriptstyle \exists E}$$

for $\forall I$, $y \equiv x$ or y not free in A, and y not free in Γ
for $\exists E$, $y \equiv x$ or y not free in A, and y not free in C or Γ.

B Examples of formal derivations in SL

B.1 Proof of $\vdash_S \{\beta \cdot \forall x \{\alpha \cdot \exists y\, P(x,y) \cdot \top\} \cdot \beta\}$ from Example 2.2

A possible derivation is:

$$\dfrac{\dfrac{\dfrac{\vdash_S \{\alpha \cdot \top \cdot \mathsf{stored}(x)\} \quad \vdash_S \{\mathsf{stored}(x) \cdot \top \cdot \mathsf{solved}(x)\}}{\vdash_S \{\alpha \cdot \top \wedge \top \cdot \mathsf{solved}(x)\}}\,{\scriptstyle \wedge_S I}}{\vdash_S \{\alpha \cdot \top \cdot \mathsf{solved}(x)\}}\,{\scriptstyle \wedge_S E_L} \qquad \vdash_S \{\mathsf{solved}(x) \cdot \exists y\, P(x,y) \cdot \top\}}{\dfrac{\dfrac{\vdash_S \{\alpha \cdot \top \wedge \exists y\, P(x,y) \cdot \top\}}{\vdash_S \{\alpha \cdot \exists y\, P(x,y) \cdot \top\}}\,{\scriptstyle \wedge_S E_L}}{\vdash_S \{\beta \cdot \forall x \{\alpha \cdot \exists y\, P(x,y) \cdot \top\} \cdot \beta\}}\,{\scriptstyle \forall_S I}}\,{\scriptstyle \wedge_S I}$$

B.2 Proof of $\vdash_S \{\top \cdot \top \cdot \mathsf{sorted}\}$ from Example 2.3

Let $\alpha := 1 \leq 2 \wedge 1 \leq 3$, and define $\mathcal{D}_1$ as

$$\dfrac{\dfrac{\vdash_S \{2 \leq 3 \wedge \alpha \cdot \top \cdot 2 \leq 3 \wedge \alpha\}}{\vdash_S \{2 \leq 3 \wedge \alpha \cdot \top \cdot \mathsf{sorted}\}}\,{\scriptstyle cons} \qquad \dfrac{\dfrac{}{\vdash_S \{3 \leq 2 \wedge \alpha \cdot \top \cdot 2 \leq 3 \wedge 1 \leq 3 \wedge 1 \leq 2\}}\,{\scriptstyle 2 \leftrightarrow 3}}{\vdash_S \{3 \leq 2 \wedge \alpha \cdot \top \cdot \mathsf{sorted}\}}\,{\scriptstyle cons}}{\vdash_S \{\alpha \cdot \top \cdot \mathsf{sorted}\}}\,{\scriptstyle cond[2 \leq 3 \vee 3 \leq 2]}$$

where for the left instance of *cons* we use $2 \leq 3 \wedge \alpha \vdash_H \mathsf{sorted}$, in the right that $2 \leq 3 \wedge 1 \leq 3 \wedge 1 \leq 2 \vdash_H \mathsf{sorted}$, and for the final instance of *cond* we use $\vdash_H 2 \leq 3 \vee 3 \leq 2$. Now let $\mathcal{D}_2$ be defined by

$$\dfrac{\dfrac{}{\vdash_S \{2 \leq 1 \wedge 2 \leq 3 \cdot \top \cdot 1 \leq 2 \wedge 1 \leq 3\}}\,{\scriptstyle 1\leftrightarrow 2} \quad \dfrac{\mathcal{D}_1}{\vdash_S \{1 \leq 2 \wedge 1 \leq 3 \cdot \top \cdot \mathsf{sorted}\}}}{\dfrac{\vdash_S \{2 \leq 1 \wedge 2 \leq 3 \cdot \top \wedge \top \cdot \mathsf{sorted}\}}{\vdash_S \{2 \leq 1 \wedge 2 \leq 3 \cdot \top \cdot \mathsf{sorted}\}}\,\wedge_S E_L}\,\wedge_S I$$

Then we have $\mathcal{D}_3$:

$$\dfrac{\dfrac{\mathcal{D}_2}{\vdash_S \{2 \leq 1 \wedge 2 \leq 3 \cdot \top \cdot \mathsf{sorted}\}} \quad \dfrac{\{1 \leq 2 \wedge 2 \leq 3 \cdot \top \cdot 1 \leq 2 \wedge 2 \leq 3\}}{\vdash_S \{1 \leq 2 \wedge 2 \leq 3 \cdot \top \cdot \mathsf{sorted}\}}\,cons}{\vdash_S \{2 \leq 3 \cdot \top \cdot \mathsf{sorted}\}}\,cond[2\leq 1 \vee 1 \leq 2]$$

where here *cond* uses $\vdash_H 2 \leq 1 \vee 1 \leq 2$, and finally

$$\dfrac{\dfrac{\vdash_S \{2 \leq 3 \cdot \top \cdot 2 \leq 3\} \quad \dfrac{}{\vdash_S \{3 \leq 2 \cdot \top \cdot 2 \leq 3\}}\,{\scriptstyle 2\leftrightarrow 3}}{\vdash_S \{\top \cdot \top \cdot 2 \leq 3\}}\,cond[2\leq 3 \vee 3 \leq 2] \quad \dfrac{\mathcal{D}_3}{\vdash_S \{2 \leq 3 \cdot \top \cdot \mathsf{sorted}\}}}{\dfrac{\vdash_S \{\top \cdot \top \wedge \top \cdot \mathsf{sorted}\}}{\vdash_S \{\top \cdot \top \cdot \mathsf{sorted}\}}\,\wedge_S E_L}\,\wedge_S I$$

C Examples of extracted terms in ST

C.1 Realizing term for $\vdash_S \{\beta \cdot \forall x \{\alpha \cdot \exists y\, P(x,y) \cdot \top\} \cdot \beta\}$ in Example 5.6

The proof term corresponding to the derivation in Appendix B.1 is as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{x : D \vdash \mathsf{write}(x) : C \quad x : D \vdash \mathsf{calc} : C}{x : D \vdash \mathsf{write}(x) \circ \mathsf{calc} : C \times C}\,\wedge_S I}{x : D \vdash \mathsf{write}(x) * \mathsf{calc} : C}\,\wedge_S E_L \quad x : D \vdash \mathsf{read} : D \times C}{x : D \vdash (\mathsf{write}(x) * \mathsf{calc}) \circ \mathsf{read} : C \times D \times C}\,\wedge_S I}{x : D \vdash (\mathsf{write}(x) * \mathsf{calc}) * \mathsf{read} : D \times C}\,\wedge_S E_L}{\vdash \lambda x\,.\,((\mathsf{write}(x) * \mathsf{calc}) * \mathsf{read}) : D \to D \times C}\,\forall_S I$$

C.2 Realizing term for $\vdash_S \{\top \cdot \top \cdot \mathsf{sorted}\}$ in Example 5.7

We construct a realizing term corresponding to the derivation given in Section B.2. First, we interpret $\mathcal{D}_1$ as

$$\dfrac{\dfrac{\vdash \mathsf{skip} : C}{\vdash \mathsf{skip} : C}\,cons \quad \dfrac{\dfrac{}{\vdash \mathsf{swap}\langle 2,3\rangle : C}\,{\scriptstyle 2\leftrightarrow 3}}{\vdash \mathsf{swap}\langle 2,3\rangle : C}\,cons}{\vdash t_1 := \mathsf{if}\,(2 \leq 3)\,\mathsf{then}\,(\mathsf{skip})\,\mathsf{else}\,(\mathsf{swap}\langle 2,3\rangle) : C}\,cond[2\leq 3 \vee 3 \leq 2]$$

and define $t_1 := \mathsf{if}\,(2 \leq 3)\,\mathsf{then}\,(\mathsf{skip})\,\mathsf{else}\,(\mathsf{swap}\langle 2,3\rangle)$. Now $\mathcal{D}_2$ is interpreted as

$$\dfrac{\dfrac{\dfrac{}{\vdash \mathsf{swap}\langle 1,2\rangle : C}\,{\scriptstyle 1\leftrightarrow 2} \quad \dfrac{\mathcal{D}_1}{\vdash t_1 : C}}{\vdash \mathsf{swap}\langle 1,2\rangle \circ t_1 : C \times C}\,\wedge_S I}{\vdash t_2 := \mathsf{swap}\langle 1,2\rangle * t_1 : C}\,\wedge_S E_L$$

where we define $t_2 := \mathsf{swap}\langle 1,2\rangle * t_1 : C$. Continuing, $\mathcal{D}_3$ is interpreted as:

$$\dfrac{\dfrac{\mathcal{D}_2}{\vdash t_2 : C} \quad \dfrac{\vdash \mathsf{skip} : C}{\vdash \mathsf{skip} : C}\,cons}{t_3 := \mathsf{if}\,(2 \leq 1)\,\mathsf{then}\,t_2\,\mathsf{else}\,(\mathsf{skip}) : C}\,cond[2\leq 1 \vee 1 \leq 2]$$

where $t_3 := \mathsf{if}\,(2 \leq 1)\,\mathsf{then}\,t_2\,\mathsf{else}\,(\mathsf{skip})$, and finally

$$\dfrac{\dfrac{\dfrac{\vdash \mathsf{skip} : C \quad \dfrac{}{\vdash \mathsf{swap}\langle 2,3\rangle : C}\,2\leftrightarrow 3}{\vdash \mathsf{if}\,(2 \leq 3)\,\mathsf{then}\,(\mathsf{skip})\,\mathsf{else}\,(\mathsf{swap}\langle 2,3\rangle) : C}\,cond[2\leq 3 \vee 3 \leq 2] \quad \dfrac{\mathcal{D}_3}{\vdash t_3 : C}}{\vdash (\mathsf{if}\,(2 \leq 3)\,\mathsf{then}\,(\mathsf{skip})\,\mathsf{else}\,(\mathsf{swap}\langle 2,3\rangle)) \circ t_3 : C \times C}\,\wedge_S I}{\vdash t := (\mathsf{if}\,(2 \leq 3)\,\mathsf{then}\,(\mathsf{skip})\,\mathsf{else}\,(\mathsf{swap}\langle 2,3\rangle)) * t_3 : C}\,\wedge_S E_L$$

D Detailed proofs

This section contains full details of (parts of) proofs that involve routine steps and calculations.

Proof of Lemma 4.8. For any variable $v : X \times Y$ we have $[p_0 v]\pi = \langle \mathsf{proj}_0 v, \pi\rangle$ and $[p_1 v]\pi = \langle \mathsf{proj}_1 v, \pi\rangle$, and we also have $[\lambda x, y\,.\,t]\pi = \langle \lambda x, \pi.\langle \lambda y.[t], \pi\rangle, \pi\rangle$. We therefore calculate

$$[(\lambda x, y.t)(p_0 v)]\pi = (\lambda x, \pi.\langle \lambda y.[t], \pi\rangle)(\mathsf{proj}_0 v)\pi = \langle \lambda y.[t][\mathsf{proj}_0 v/x], \pi\rangle$$

and thus

$$[(\lambda x, y.t)(p_0 v)(p_1 v)]\pi = (\lambda y.[t][\mathsf{proj}_0 v/x])(\mathsf{proj}_1 v)\pi = [t][\mathsf{proj}_0 v/x, \mathsf{proj}_1 v/y]\pi$$

Finally, we can see that if $\langle a, b, \pi_1\rangle := [s]\pi$ then

$$\begin{aligned}[(\lambda^* v.t)(s)]\pi &= (\lambda v.[(\lambda x, y.t)(p_0 v)(p_1 v)])(\langle a, b\rangle)\pi_1 \\ &= (\lambda v.[t][\mathsf{proj}_0 v/x, \mathsf{proj}_1 v/y])(\langle a, b\rangle)\pi_1 \\ &= [t][\mathsf{proj}_0 v/x, \mathsf{proj}_1 v/y][\langle a, b\rangle/v]\pi_1 \\ &= [t][a/x, b/y]\pi_1\end{aligned}$$

□

Straightforward cases for Theorem 5.4.

- For the axiom $\Gamma \vdash_S \{\alpha \cdot A \cdot \alpha\}$, if $A^u \in \Gamma$ we define $t := x$ for the corresponding variable $x : \tau_S(A)$. Then $[x]\pi := \langle x, \pi\rangle$ for x sr A and $[\alpha](\pi)$. For $\Gamma \vdash_S \{\alpha \cdot \top \cdot \alpha\}$ we define $t := \mathsf{skip}$ and the verification is even simpler.

- ($\wedge_S I$) Given terms s, t with $[s]$ sr $\{\alpha \cdot A \cdot \beta\}$ and $[t]$ sr $\{\beta \cdot B \cdot \gamma\}$, from $[\alpha](\pi)$ we can infer a sr A and $[\beta](\pi_1)$ for $\langle a, \pi_1\rangle := [s]\pi$, and from $[\beta](\pi_1)$ it follows that b sr B and $[\gamma](\pi_2)$ for $\langle b, \pi_2\rangle := [t]\pi_1$, therefore we have shown that $[s \circ t]$ sr $\{\alpha \cdot A \wedge B \cdot \beta\}$.

- ($\wedge_S E_i$) If $[t]$ sr $\{\alpha \cdot A \wedge B \cdot \beta\}$ then $\langle a, b\rangle$ sr $A \wedge B$ and $[\beta](\pi_1)$ follow from $[\alpha](\pi)$, where $\langle a, b, \pi_1\rangle := [t]\pi$. But then $[p_0 t]$ sr $\{\alpha \cdot A \cdot \beta\}$ and $[p_1 t]$ sr $\{\alpha \cdot B \cdot \beta\}$.

- ($\vee_S I_i$) If $[t]$ sr $\{\alpha \cdot A \cdot \beta\}$ and $[\alpha](\pi)$ holds, then a sr A and $[\beta](\pi_1)$ for $\langle a, \pi_1\rangle := [t]\pi$, and therefore

  $$(b = \mathsf{f} \Rightarrow a \text{ sr } A) \wedge (b = \mathsf{t} \Rightarrow 0_{\tau_S(B)} \text{ sr } B)$$

  for $b := \mathsf{f}$. Thus $[\iota_0 t]$ sr $A \vee B$. By an entirely analogous argument we can show that $[\iota_1 t]$ sr $A \vee B$ whenever $[t]$ sr B.

- ($\Rightarrow_S I$) If $t(x)$ is such that $[t](x)$ sr $\{\alpha \cdot B \cdot \beta\}$ whenever x sr A, then by definition we have $\lambda x.[t]$ sr $A \Rightarrow \{\alpha \cdot B \cdot \beta\}$, and therefore $[\lambda x.t]$ sr $\{\gamma \cdot A \Rightarrow \{\alpha \cdot B \cdot \beta\} \cdot \gamma\}$ for any γ.

- ($\Rightarrow_S E$) Assume that $[s]$ sr $\{\beta \cdot A \cdot \gamma\}$ and $[t]$ sr $\{\alpha \cdot A \Rightarrow \{\gamma \cdot B \cdot \delta\} \cdot \beta\}$. If $[\alpha](\pi)$ holds then defining $\langle f, \pi_1 \rangle := [t]\pi$ we have $[\beta]\pi_1$ and
$$x \text{ sr } A \Rightarrow fx \text{ sr } \{\gamma \cdot B \cdot \delta\}$$
Similarly, defining $\langle a, \pi_2 \rangle := [s]\pi_1$, it follows that $[\gamma](\pi_2)$ and a sr A. Finally, setting $\langle b, \pi_3 \rangle := fa\pi_2$ it follows that b sr B and $[\delta](\pi_3)$, and we have therefore proven that $[ts]$ sr $\{\alpha \cdot B \cdot \delta\}$.

- ($\bot_S E$) If $[t]$ sr $\{\alpha \cdot \bot \cdot \beta\}$ then from $[\alpha](\pi)$ we can infer a sr $\bot$ and $[\beta](\pi_2)$ for $\langle a, \pi_1 \rangle := [t]\pi$. But a sr $\bot = \bot$, and from $\bot$ we can deduce anything, and in particular $0_{\tau_S(A)}$ sr A and $[\gamma](\pi)$, from which it follows that $[\mathsf{default}_{\tau_S(A)}]$ sr $\{\alpha \cdot A \cdot \gamma\}$.

- ($\forall_S I$) Suppose that $t(x)$ is such that $[t](y)$ sr $\{\alpha[y/x] \cdot A[y/x] \cdot \beta[y/x]\}$, where $y \equiv x$ or y is not free in $\{\alpha \cdot A \cdot \beta\}$, and y is not free in Γ. Then since y is not free in any of the assumptions x_i sr A_i, we can deduce in S^ω that
$$\forall x^D\, [t](x) \text{ sr } \{\alpha \cdot A \cdot \beta\}$$
and therefore $\lambda x.[t]$ sr $\forall x\, \{\alpha \cdot A \cdot \beta\}$, and thus (just as for $\Rightarrow_S I$) we have
$$[\lambda x.t] \text{ sr } \{\gamma \cdot \forall x\, \{\alpha \cdot A \cdot \beta\} \cdot \gamma\}$$
for any γ.

- ($\forall_S E$) Suppose that $[s]$ sr $\{\alpha \cdot \forall x\, \{\beta \cdot A \cdot \gamma\} \cdot \beta[t/x]\}$ and that $[\alpha](\pi)$ holds. Then we have f sr $\forall x\, \{\beta \cdot A \cdot \gamma\}$ and $[\beta][t/x](\pi_1)$ for $\langle f, \pi \rangle := [s]\pi$. Now, using Lemma 4.7 we have $[st]\pi = ft\pi_1$ for the natural interpretation of t in S^ω, since we can prove in S^ω that
$$ft \text{ sr } \{\beta[t/x] \cdot A[t/x] \cdot \gamma[t/x]\}$$
it follows that a sr $A[t/x]$ and $[\gamma][t/x](\pi_2)$ for $\langle a, \pi_2 \rangle := ft\pi_1$, and therefore we have shown that $[st]$ sr $\{\alpha \cdot A[t/x] \cdot \gamma[t/x]\}$.

- ($\exists_S I$) If $[s]$ sr $\{\alpha \cdot A[t/x] \cdot \beta\}$ and $[\alpha](\pi)$ then a sr $A[t/x]$ and $[\beta](\pi_1)$ for $\langle a, \pi_1 \rangle := [s]\pi$. By Lemma 5.3 we therefore have $(a \text{ sr } A)[t/x]$, and therefore $\langle t, a \rangle$ sr $\exists x\, A$. Observing (using Lemma 4.7) that $[t \circ s]\pi = \langle t, a, \pi_1 \rangle$, we have shown that $[t \circ s]$ sr $\{\alpha \cdot \exists x\, A \cdot \beta\}$.

- (*cons*) If $\alpha \vdash_H \beta$ and $\gamma \vdash_H \delta$ then by Lemma 4.3 both $[\alpha](\pi) \Rightarrow [\beta](\pi)$ and $[\gamma](\pi) \Rightarrow [\delta](\pi)$ are provable in S^ω (respectively $S^\omega + \Lambda_{S^\omega}$ for the general version of the theorem) for any $\pi : S$. It is then easy to show that if $[t]$ sr $\{\beta \cdot A \cdot \gamma\}$ then we also have $[t]$ sr $\{\alpha \cdot A \cdot \delta\}$.

□