JFP **22** (4–5): 477–528, 2012. © Cambridge University Press 2012
doi:10.1017/S095679681200024X

The impact of higher-order state and control effects on local relational reasoning

DEREK DREYER and GEORG NEIS
Max Planck Institute for Software Systems (MPI-SWS)
(*e-mail:* {dreyer, neis}@mpi-sws.org)

LARS BIRKEDAL
IT University of Copenhagen
(*e-mail:* birkedal@itu.dk)

Abstract

Reasoning about program equivalence is one of the oldest problems in semantics. In recent years, useful techniques have been developed, based on bisimulations and logical relations, for reasoning about equivalence in the setting of increasingly realistic languages—languages nearly as complex as ML or Haskell. Much of the recent work in this direction has considered the interesting representation independence principles *enabled* by the use of local state, but it is also important to understand the principles that powerful features like higher-order state and control effects *disable*. This latter topic has been broached extensively within the framework of game semantics, resulting in what Abramsky dubbed the "semantic cube": fully abstract game-semantic characterizations of various axes in the design space of ML-like languages. But when it comes to reasoning about many actual examples, game semantics does not yet supply a useful technique for proving equivalences.

In this paper, we marry the aspirations of the semantic cube to the powerful proof method of *step-indexed Kripke logical relations*. Building on recent work of Ahmed *et al.* (2009), we define the first fully abstract logical relation for an ML-like language with recursive types, abstract types, general references and call/cc. We then show how, under orthogonal restrictions to the expressive power of our language—namely, the restriction to first-order state and/or the removal of call/cc—we can enhance the proving power of our possible-worlds model in correspondingly orthogonal ways, and we demonstrate this proving power on a range of interesting examples. Central to our story is the use of *state transition systems* to model the way in which properties of local state evolve over time.

1 Introduction

Reasoning about program equivalence is one of the oldest problems in semantics, with applications to program verification ("Is an optimized program equivalent to some reference implementation?"), compiler correctness ("Does a program transformation preserve the semantics of the source program?"), representation independence ("Can we modify the internal representation of an abstract data type without affecting the behavior of clients?"), and more besides.

The canonical notion of program equivalence for many applications is *observational* (or *contextual*) equivalence (Morris, 1968). Two programs are observationally equivalent if no program context can distinguish them by getting them to exhibit observably different input/output behavior. Reasoning about observational equivalence directly is difficult, due to the universal quantification over program contexts. Consequently, there has been a huge amount of work on developing useful models and logics for observational equivalence, and in recent years this line of work has scaled to handle increasingly realistic languages—languages nearly as complex as ML or Haskell, with features like general recursive types, general (higher-order) mutable references, and first-class continuations.

The focus of much of this recent work—e.g., environmental bisimulations (Koutavas & Wand, 2006; Sumii & Pierce, 2007; Sumii, 2009; Sangiorgi *et al.*, 2011), normal form bisimulations (Støvring & Lassen, 2007; Koutavas & Lassen, 2008), step-indexed Kripke logical relations (Appel & McAllester, 2001; Ahmed, 2004; Ahmed *et al.*, 2009)—has been on establishing some effective techniques for reasoning about programs that actually *use* the interesting, semantically complex features (state, continuations, etc.) of the languages being modeled. For instance, most of the work on languages with state concerns the various kinds of representation independence principles that arise due to the use of *local state* as an abstraction mechanism.

But of course this is only part of the story. When features are added to a language, they also enrich the expressive power of program *contexts*. Hence, programs that do *not* use those new features, and that are observationally equivalent in the absence of those features, might not be observationally equivalent in their presence. One well-known example of this is the loss of referential transparency in an impure language like ML. Another shows up in the work of Johann & Voigtländer (2006), who study the negative impact that Haskell's strictness operator `seq` has on the validity of short-cut fusion and other free-theorems-based program transformations. In our case, we are interested in relational reasoning about *stateful* programs, so we will be taking a language with some form of mutable state as our baseline. Nonetheless, we feel it is important not only to study the kinds of local reasoning principles that stateful programming can *enable*, but also to understand the principles that powerful features like higher-order state and control effects *disable*.

This latter topic has been broached extensively within the framework of *game semantics*. In the 1990s, Abramsky set forth a research program (subsequently undertaken by a number of people) concerning what he called the *semantic cube* (Laird, 1997; Abramsky *et al.*, 1998; Murawski, 2005). The idea was to develop fully abstract game-semantic characterizations of various axes in the design space of ML-like languages. For instance, the absence of mutable state can be modeled by restricting game strategies to be *innocent*, and the absence of control operators can be modeled by restricting game strategies to be *well-bracketed*. These restrictions are orthogonal to one another and can be composed to form fully abstract models of languages with different combinations of effects. Unfortunately, when it comes to reasoning about many actual examples, these game-semantics models do not yet supply a useful technique for proving programs equivalent, except in fairly restricted languages (see Section 10 for further discussion).

One possible reason for the comparative lack of attention paid to this issue in the setting of relational reasoning is that some key techniques that have been developed for reasoning about local state—notably, Pitts and Stark's method of *local invariants* (Pitts & Stark, 1998)—turn out to work just as well in a language with higher-order state and call/cc as they do in the simpler setting (first-order state, no control operators) in which they were originally proposed. Before one can observe the negative impact of certain language features on relational reasoning principles, one must first develop a proof technique that actually *exploits* the absence of those features.

1.1 Overview

In this paper, we marry the aspirations of Abramsky's semantic cube to the powerful proof method of *step-indexed Kripke logical relations*. Specifically, we show how to define a fully abstract logical relation for an ML-like language with recursive types, abstract types, general references and call/cc. Then, we show how, under orthogonal restrictions to the expressive power of our language—namely, the restriction to first-order state and/or the removal of call/cc—we can enhance the proving power of our model in correspondingly orthogonal ways, and we demonstrate this power on a range of interesting examples.

Our work builds closely on that of Ahmed *et al.* (2009) (hereafter, ADR), who gave the first logical relation for modeling a language with both abstract types and higher-order state. We take ADR as a starting point because the concepts underlying that model provide a rich framework in which to explore the impact of various computational effects on relational reasoning. In particular, one of ADR's main contributions was an extension of Pitts and Stark's aforementioned "local invariants" method with the ability to establish properties about local state that *evolve* over time in some controlled fashion. ADR exploited this ability in order to reason about *generative* (or *state-dependent*) ADTs.

The central contribution of our present paper is to observe that the degree of freedom with which local state properties may evolve depends directly on which particular effects are present in the programming language under consideration. In order to expound this observation, we first recast the ADR model in the more familiar terms of *state transition systems* (Section 3). The basic idea is that the "possible worlds" of the ADR model are really state transition systems, wherein each state dictates a potentially different property about the heap, and the transitions between states control how the heap properties are allowed to evolve. Aside from being somewhat simpler than ADR's formulation of possible worlds (which relied on various non-standard anthropomorphic notions like "populations" and "laws"), our formulation highlights the essential notion of a *state transition*, which plays a crucial role in our story.

Next, in Section 4, we explain how to extend the ADR model with support for first-class continuations via the well-studied technique of *biorthogonality* (aka $\top\top$-closure) (Krivine, 1994; Pitts & Stark, 1998). The technical details of this extension are fairly straightforward, with the use of biorthogonality turning out to be completely orthogonal (no pun intended) to the other advanced aspects of the ADR model. That said, this is to our knowledge *the first logical-relations model for a language with call/cc and state.* Moreover, a side benefit of biorthogonality is that it renders our model *both sound and complete* with respect to observational equivalence (unlike ADR's, which was only sound).[1] Interestingly, nearly all of the example program equivalences proved in the ADR paper continue to hold in the presence of call/cc, and their proofs carry over easily to our present formulation. (There is one odd exception, the "callback with lock" example, for which the ADR proof was very fiddly and *ad hoc*. We investigate this example in great detail, as we describe below.)

The ADR paper also included several interesting examples that their method was *unable* to handle. The unifying theme of these examples is that they rely on the *well-bracketed* nature of computation—i.e., the assumption that control flow follows a stack-like discipline—an assumption that is only valid in the *absence* of call/cc. In Section 5, we consider two simple but novel enhancements to our state-transition-system model—*private transitions* and *inconsistent states*—which are only sound in the absence of call/cc and which correspondingly enable us to prove all of ADR's "well-bracketed examples".

Conversely, in Section 6, we consider the additional reasoning power gained by restricting the language to first-order state. We observe that this restriction enables *backtracking* within a state transition system, and we demonstrate the utility of this feature on several examples.

The above extensions to our basic state-transition-system model are orthogonal to each other, and can be used independently or in combination. One notable example of this is ADR's "callback with lock" equivalence (mentioned above), an equivalence that holds *in the presence of either* higher-order state or call/cc but not both. Using private transitions but no backtracking, we can prove this equivalence in the presence of higher-order state but no call/cc; and using backtracking but no private transitions, we can prove it in the presence of call/cc but only first-order state. Yet another well-known example, due originally to O'Hearn & Reddy (1995), is true only *in the absence of both* higher-order state and call/cc; hence, it should come as no surprise that our novel proof of this example (presented in detail in Section 9.2) involves all three of our model's new features working in tandem.

Most of the paper is presented in an informal, pedagogical style. Indeed, one advantage of our state transition systems is that they lend themselves to clean "visual" proof sketches. In Section 7, we make our proof method formally precise and present some of the key metatheoretic results. We sketch some interesting parts of their proofs, but the full details can be found in the companion technical appendix (Dreyer *et al.*, 2012).

In Section 8, we consider how our Kripke logical relations are affected by the addition of *exceptions* to the language. Unlike call/cc, exceptions do not impose restrictions on our state transition systems, but they do require us to account for exceptional behavior in our proofs.

[1] It is important to note that the completeness result has nothing to do with the particular features present in the language, and all to do with the use of biorthogonality. In particular, biorthogonality gives us a uniform way of constructing fully abstract models for *all* of the different languages considered in this paper, regardless of whether they contain call/cc, general references, etc. See Section 10 for further discussion of this point.

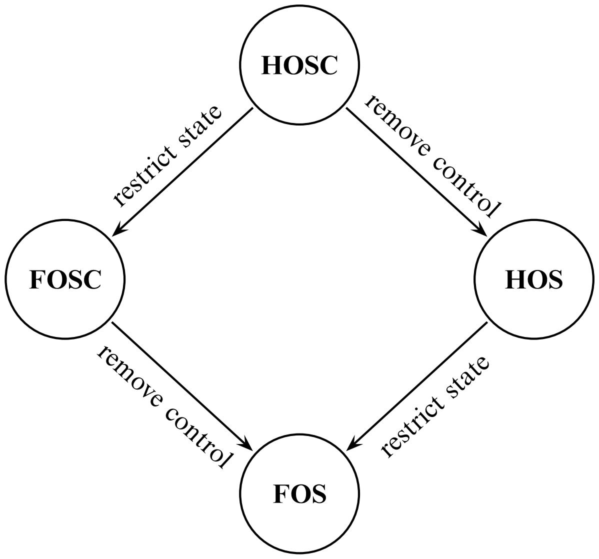
In Section 9, we work through the proofs of several challenging equivalences that hold in the presence or absence of different features, thus demonstrating the power of the various extensions to our state-transition-system model. Even more examples can be found in Dreyer *et al.* (2012).

Finally, in Section 10, we compare our methods to related work and suggest some directions for future work.

2 The language(s) under consideration

In its unrestricted form, the language that we consider is a standard polymorphic lambda calculus with existential, pair, and iso-recursive types, general references (higher-order state), and first-class continuations (call/cc). We call this language **HOSC**. Its syntax and excerpts of its typing rules ($\Sigma;\Delta;\Gamma \vdash e : \tau$) and call-by-value semantics ($\langle h;e\rangle \hookrightarrow \langle h';e'\rangle$) are given in Figure 1. Dots ($\dots$) in the syntax cover primitive operations on base types b, such as addition and if-then-else. To ensure unique typing, various constructs have explicit type annotations, which we will typically omit if they are implicit from context. Evaluation contexts K, injected into the term language via $\mathsf{cont}_\tau\ K$, represent first-class continuations. They are a subset of general contexts C ("terms with a hole"), which are not shown here, but are standard. Their typing judgment $\vdash C : (\Sigma;\Delta;\Gamma;\tau) \rightsquigarrow (\Sigma';\Delta';\Gamma';\tau')$ basically says that for any e with $\Sigma;\Delta;\Gamma \vdash e : \tau$ we have $\Sigma';\Delta';\Gamma' \vdash C[e] : \tau'$. The continuation typing judgment $\Sigma;\Delta;\Gamma \vdash K \div \tau$ says that K is an evaluation context with a hole of type τ. Finally, contextual (or observational) approximation, written $\Sigma;\Delta;\Gamma \vdash e_1 \lesssim_{\mathrm{ctx}} e_2 : \tau$, means that in any well-typed program context C, if $C[e_1]$ terminates, then so does $C[e_2]$. Contextual (or observational) equivalence is then defined as approximation in both directions.

By restricting **HOSC** in two orthogonal ways, we obtain three fragments of interest:



$$\tau ::= \alpha \mid b \mid \tau_1 \times \tau_2 \mid \tau_1 \to \tau_2 \mid \forall\alpha.\,\tau \mid \exists\alpha.\,\tau \mid \mu\alpha.\,\tau \mid \mathsf{ref}\ \tau \mid \mathsf{cont}\ \tau$$

$$\begin{aligned} e ::=\ & x \mid l \mid \langle e_1, e_2\rangle \mid e.1 \mid e.2 \mid \lambda x{:}\tau.e \mid e_1\ e_2 \mid \Lambda\alpha.e \mid e\ \tau \mid \mathsf{pack}\ \langle\tau_1, e\rangle\ \mathsf{as}\ \tau_2 \mid \\ & \mathsf{unpack}\ e_1\ \mathsf{as}\ \langle\alpha, x\rangle\ \mathsf{in}\ e_2 \mid \mathsf{roll}_\tau\ e \mid \mathsf{unroll}\ e \mid \mathsf{ref}\ e \mid e_1 := e_2 \mid !e \mid e_1 == e_2 \mid \mathsf{cont}_\tau\ K \mid \\ & \mathsf{call/cc}_\tau\,(x.\ e) \mid \mathsf{throw}_\tau\ e_1\ \mathsf{to}\ e_2 \mid \ldots \end{aligned}$$

$$\begin{aligned} K ::=\ & \bullet \mid \langle K, e_2\rangle \mid \langle v_1, K\rangle \mid K.1 \mid K.2 \mid K\ e_2 \mid v_1\ K \mid K\ \tau \mid \mathsf{pack}\ \langle\tau_1, K\rangle\ \mathsf{as}\ \tau_2 \mid \\ & \mathsf{unpack}\ K\ \mathsf{as}\ \langle\alpha, x\rangle\ \mathsf{in}\ e_2 \mid \mathsf{roll}_\tau\ K \mid \mathsf{unroll}\ K \mid \mathsf{ref}\ K \mid K := e_2 \mid v_1 := K \mid !K \mid K == e_2 \mid \\ & v_1 == K \mid \mathsf{throw}_\tau\ K\ \mathsf{to}\ e_2 \mid \mathsf{throw}_\tau\ v_1\ \mathsf{to}\ K \mid \ldots \end{aligned}$$

$$v ::= x \mid l \mid \langle v_1, v_2\rangle \mid \lambda x{:}\tau.e \mid \Lambda\alpha.e \mid \mathsf{pack}\ \langle\tau_1, v\rangle\ \mathsf{as}\ \tau_2 \mid \mathsf{roll}_\tau\ v \mid \mathsf{cont}_\tau\ K \mid \ldots$$

$$\begin{array}{lll} \ldots & & \\ \langle h; K[\mathsf{ref}\ v]\rangle & \hookrightarrow \langle h \uplus \{l \mapsto v\}; K[l]\rangle & (l \notin \mathrm{dom}(h)) \\ \langle h; K[l := v]\rangle & \hookrightarrow \langle h[l \mapsto v]; K[\langle\rangle]\rangle & (l \in \mathrm{dom}(h)) \\ \langle h; K[!l]\rangle & \hookrightarrow \langle h; K[v]\rangle & (h(l) = v) \\ \langle h; K[l_1 == l_2]\rangle & \hookrightarrow \langle h; K[\mathsf{tt}]\rangle & (l_1 = l_2) \\ \langle h; K[l_1 == l_2]\rangle & \hookrightarrow \langle h; K[\mathsf{ff}]\rangle & (l_1 \neq l_2) \\ \langle h; K[\mathsf{call/cc}_\tau\,(x.\ e)]\rangle & \hookrightarrow \langle h; K[e[\mathsf{cont}_\tau\ K/x]]\rangle & \\ \langle h; K[\mathsf{throw}_\tau\ v\ \mathsf{to}\ \mathsf{cont}_{\tau'}\ K']\rangle & \hookrightarrow \langle h; K'[v]\rangle & \end{array}$$

$$\begin{array}{llll} \text{Heap typings} & \Sigma & ::= & \cdot \mid \Sigma, l{:}\tau \quad \text{where } \mathrm{fv}(\tau) = \emptyset \\ \text{Type environments} & \Delta & ::= & \cdot \mid \Delta, \alpha \\ \text{Term environments} & \Gamma & ::= & \cdot \mid \Gamma, x{:}\tau \end{array}$$

$$\frac{l{:}\tau \in \Sigma}{\Sigma;\Delta;\Gamma \vdash l : \mathsf{ref}\ \tau} \qquad \frac{\Sigma;\Delta;\Gamma \vdash e : \tau}{\Sigma;\Delta;\Gamma \vdash \mathsf{ref}\ e : \mathsf{ref}\ \tau} \qquad \frac{\Sigma;\Delta;\Gamma \vdash e_1 : \mathsf{ref}\ \tau \quad \Sigma;\Delta;\Gamma \vdash e_2 : \tau}{\Sigma;\Delta;\Gamma \vdash e_1 := e_2 : \mathsf{unit}}$$

$$\frac{\Sigma;\Delta;\Gamma \vdash e : \mathsf{ref}\ \tau}{\Sigma;\Delta;\Gamma \vdash !e : \tau} \qquad \frac{\Sigma;\Delta;\Gamma \vdash e_1 : \mathsf{ref}\ \tau \quad \Sigma;\Delta;\Gamma \vdash e_2 : \mathsf{ref}\ \tau}{\Sigma;\Delta;\Gamma \vdash e_1 == e_2 : \mathsf{bool}}$$

$$\frac{\forall l{:}\tau \in \Sigma.\ \Sigma;\cdot;\cdot \vdash h(l) : \tau}{\vdash h : \Sigma}$$

$$\frac{\vdash K : (\Sigma;\Delta;\Gamma;\tau) \rightsquigarrow (\Sigma;\Delta;\Gamma;\tau')}{\Sigma;\Delta;\Gamma \vdash K \div \tau} \qquad \frac{\Sigma;\Delta;\Gamma \vdash K \div \tau}{\Sigma;\Delta;\Gamma \vdash \mathsf{cont}_\tau\ K : \mathsf{cont}\ \tau}$$

$$\frac{\Sigma;\Delta;\Gamma, x{:}\mathsf{cont}\ \tau \vdash e : \tau}{\Sigma;\Delta;\Gamma \vdash \mathsf{call/cc}_\tau\,(x.\ e) : \tau} \qquad \frac{\Sigma;\Delta;\Gamma \vdash e' : \tau' \quad \Sigma;\Delta;\Gamma \vdash e : \mathsf{cont}\ \tau'}{\Sigma;\Delta;\Gamma \vdash \mathsf{throw}_\tau\ e'\ \mathsf{to}\ e : \tau}$$

$$\boxed{\Sigma;\Delta;\Gamma \vdash e_1 \precsim_{\mathrm{ctx}} e_2 : \tau} \overset{\mathrm{def}}{=} \begin{aligned}[t] & \Sigma;\Delta;\Gamma \vdash e_1 : \tau \wedge \Sigma;\Delta;\Gamma \vdash e_2 : \tau \wedge \forall C, \Sigma', \tau', h. \\ & \vdash C : (\Sigma;\Delta;\Gamma;\tau) \rightsquigarrow (\Sigma';\cdot;\cdot;\tau') \wedge \vdash h : \Sigma' \wedge \\ & \langle h; C[e_1]\rangle\downarrow \Rightarrow \langle h; C[e_2]\rangle\downarrow \end{aligned}$$

Fig. 1. The language **HOSC**.

FOSC The result of restricting to first-order state. Concretely, this means only permitting reference types $\mathsf{ref}\, b$, where b represents base types like int, bool, etc.

HOS The result of removing call/cc, i.e., dropping the type $\mathsf{cont}\,\tau$ and the corresponding three term-level constructs.

FOS The result of making both of the above restrictions.

3 A model based on state transition systems

The Ahmed–Dreyer–Rossberg (ADR) model (Ahmed *et al.*, 2009), on which our model is based, is a step-indexed Kripke logical relation for the language **HOS**. In this section, we will briefly review what a step-indexed Kripke logical relation is, what is interesting about the ADR model, and how we can recast the essence of the ADR model in terms of *state transition systems.*

Step-indexed Kripke logical relations. Logical relations are one of the best-known methods for local reasoning about equivalence (or, more generally, approximation) in higher-order, typed languages. The basic idea is to define the equivalence or approximation relation in question inductively over the type structure of the language, with each type constructor being interpreted by the logical connective to which it corresponds. For instance, two functions are logically related if relatedness of their arguments *implies* relatedness of their results; two existential packages are logically related if there *exists* a relational interpretation of their hidden type representations that is preserved by their operations; and so forth.

In order to reason about equivalence in the presence of state, it becomes necessary to place constraints on the heaps under which programs are evaluated. This is where *Kripke* logical relations come in. Kripke logical relations (Pitts & Stark, 1998) are logical relations indexed by a *possible world* W, which codifies some set of heap constraints. Roughly speaking, e_1 is related to e_2 under W only if they behave "the same" when run under any heaps h_1 and h_2 that satisfy the constraints of W. When reasoning about programs that maintain some *local* state, possible worlds allow us to impose whatever invariants on the local state we want, so long as we ensure that those invariants are preserved by the code that accesses the state.

To make things concrete, consider the following example:

$$\begin{aligned}
\tau &= (\mathsf{unit} \to \mathsf{unit}) \to \mathsf{int}\\
e_1 &= \mathsf{let}\ x = \mathsf{ref}\ 1\ \mathsf{in}\ \lambda f.(f\ \langle\rangle; !x)\\
e_2 &= \lambda f.(f\ \langle\rangle; 1)
\end{aligned}$$

We would like to show that e_1 and e_2 are observationally equivalent at type τ. The reason, intuitively, is obvious: the reference x is kept private (i.e., it is never leaked to the context), and since it is never modified by the function returned by e_1, it will always point to 1. To prove this using Kripke logical relations, we would set out to prove that e_1 and e_2 are related under an arbitrary initial world W. So suppose we evaluate the two terms under heaps h_1 and h_2 that satisfy W. Since the evaluation of e_1 results in the allocation of some *fresh* memory location for x (i.e., $x \notin \mathrm{dom}(h_1)$), we know that the initial world W cannot already contain any constraints governing the contents of x. (If it contained such a constraint, h_1 would have had to satisfy it, and hence x would have to be in $\mathrm{dom}(h_1)$.) So we may extend W with a *new* invariant stating that $x \hookrightarrow 1$ (i.e., x points to 1). It then remains to show that the two λ-abstractions are logically related under this extended world—i.e., under the assumption that $x \hookrightarrow 1$—which is straightforward.

Finally, *step-indexed* logical relations (Appel & McAllester, 2001; Ahmed, 2004) were proposed (originally by Appel and McAllester) as a way to account for semantically problematic features, such as general recursive types, whose relational interpretations are seemingly "cyclic" and thus difficult to define inductively. The idea is simply to stratify the construction of the logical relation by a natural number (or "step index"), representing roughly the number of steps of computation for which the programs in question behave in a related manner.

One of the key contributions of the ADR model was to combine the machinery of step-indexed logical relations with that of Kripke logical relations in order to model higher-order state. While the details of this construction are quite interesting, they are orthogonal to the novel contributions of the model we present in this paper. Indeed, our present model follows the ADR model very closely in its use of step-indexing to resolve circularities in the construction, and so we refer the interested reader to the ADR paper for details.

ADR and state transition systems. The other key contribution of the ADR model was to provide an enhanced notion of possible world, which has the potential to express properties of local state that *evolve* over time. To motivate this feature of ADR, consider a simple variant of the example shown above, in which the first program e_1 is replaced by

$$e_1 = \mathsf{let}\ x = \mathsf{ref}\ 0\ \mathsf{in}\ \lambda f.\,(x := 1; f\ \langle\rangle; !x)$$

Here, x starts out pointing to 0, but if the function that e_1 evaluates to is ever called, x will be set to 1 and will never change back to 0. In this case, the only interesting invariant one can prove about x is that it points to *either* 0 or 1, but this invariant is insufficient to establish that after the call to the callback f, the contents of x have not changed back to 0. Pitts & Stark (1998) called this example the "awkward" example, and they could not prove it because their possible-worlds model only supported heap *invariants*.

While the awkward example is clearly contrived, it is also a minimal representative of a useful class of programs in which changes to local state occur in some monotonic fashion. As ADR showed, this includes well-known *generative* (or *state-dependent*) ADTs, in which the interpretation of an abstract type grows over time in correspondence with changes to some local state.

ADR's solution was to generalize possible worlds' notion of "heap constraint" to express heap properties that change in a controlled fashion. We can understand their possible worlds as essentially *state transition systems*, where each state determines a particular heap property, and where the transitions determine how the heap property may evolve. For instance, in the case of the awkward example, ADR would represent the heap constraint on x via the following state transition system (STS):

$$\begin{tikzcd} \boxed{x \hookrightarrow 0} \arrow[r] & \boxed{x \hookrightarrow 1} \end{tikzcd}$$

Initially, x points to 0, and then it is set to 1. Since the call to the callback f occurs when we are in the $x \hookrightarrow 1$ state, we know it must return in the same state since there is no transition out of that state. Correspondingly, it is necessary to also show that the $x \hookrightarrow 1$ state is really final—i.e., if the function to which e_1 evaluates is called in that state, it will not change x's contents again—but this is obvious.

In ADR, states are called "populations" and state transition systems are called "laws," but the power of their possible worlds is very similar to that of our STSs (as we have described them thus far), and most of their proofs are straightforwardly presentable in terms of STSs. That said, the two models are not identical. In particular, there is one example we are aware of, the "callback with lock" example, that is provable in the ADR model but not in our basic STS model. As we will see shortly, there are good reasons why this example is not provable in our basic STS model, and in Section 5.1, we will show how to extend our STSs in order to prove this very example in a much simpler, cleaner way than the ADR model does.

4 Biorthogonality, call/cc, and full abstraction

One point on which different formulations of Kripke logical relations differ is the precise formulation of the logical relation for *terms*. The ADR model employs a "direct-style" term relation, which can be described informally as follows: two terms e_1 and e_2 are logically related under world W iff whenever they are evaluated in initial heaps h_1 and h_2 satisfying W, they either both diverge or they both converge to machine configurations $\langle h_1'; v_1\rangle$ and $\langle h_2'; v_2\rangle$ such that h_1' and h_2' satisfy W' and v_1 and v_2 are logically related values under W', where W' is *some* "future world" of W. (By "future world," we mean that W' extends W with new constraints about freshly allocated pieces of the heap, and/or the heap constraints of W may have evolved to different heap constraints in W' according to the STSs in W.) We call this a direct-style term relation because it involves evaluating the terms *directly* to values and then showing relatedness of those values in some future world.

An alternative approach, first employed in the logical relations setting by Pitts & Stark (1998) but subsequently adopted by several others (e.g., Johann, 2003; Bohr, 2007; Benton & Hur, 2009), is what one might call a "CPS" term relation, although it is more commonly known as a *biorthogonal* (or $\top\top$-closed) term relation. The idea is to define two terms to be related under world W if they co-terminate (both converge or both diverge) when evaluated under heaps that satisfy W *and under continuations* K_1 *and* K_2 *related under* W. The latter (continuation relatedness) is then defined to mean that, for any future world W' of W, the continuations K_1 and K_2 co-terminate when applied (under heaps that satisfy W') to *values* that are related under W'. In this way, the logical relation for values is lifted to a logical relation for terms by a kind of CPS transform.

The main arguable advantage of the direct-style term relation is that its definition is perhaps more intuitive, corresponding closely to the proof sketches of the sort that we will present informally in the sections that follow. That said, in any language for which a direct-style relation is sound, it is typically possible to start instead with a biorthogonal relation and then prove a direct-style proof principle—e.g., Pitts and Stark's "principle of local invariants" (Pitts & Stark, 1998)—as a corollary.

The advantages of the biorthogonal approach are clearer. First, it automagically renders the logical relation *complete* with respect to observational equivalence, largely irrespective of the particular features in the language under consideration. (Actually, it is not so magical: $\top\top$-closure is essentially a kind of closure under observational equivalence.) Second, and perhaps more importantly, the biorthogonal approach scales to handle languages with first-class continuations, such as our **HOSC** and **FOSC**, while the direct-style does not. The reason for this is simple: the direct-style approach is only sound if the evaluation of terms is *independent* of the continuation under which they are evaluated. If the terms' behavior is context-dependent, then it does not suffice to consider their co-termination under the empty continuation, which is effectively what the direct-style term relation does. Rather, it becomes necessary to consider co-termination of whole programs (terms together with their continuations), as the biorthogonal relation does.

Thus, in this paper we adopt the biorthogonal approach. This enables us to easily adapt all the proofs from the ADR paper (save for one) to also work for a language with call/cc. (The one exception is the "callback with lock" equivalence, which simply does not hold in the presence of call/cc.) Additionally, we can prove equivalences involving programs that manipulate *both* call/cc and higher-order state. One well-known challenging example of such an equivalence is the correctness of Friedman and Haynes' encoding of call/cc via "one-shot" continuations (continuations that can only be invoked once) (Friedman & Haynes, 1985; Støvring & Lassen, 2007). The basic idea of the encoding is to model an unrestricted continuation using a private (local) ref cell that contains a one-shot continuation. Every time the continuation is invoked, the ref cell is updated with a fresh one-shot continuation. With biorthogonal logical relations, the proof of this example is completely straightforward, employing just a simple invariant on the private ref cell. As far as we know, though, this proof is novel. Full details are given in Section 9.

It is worth noting that, although the kinds of example programs we focus on in this paper do not involve abstract or recursive types, a number of the ADR examples do. Therefore, in the new models we present in this paper, we will include support for these features, in order to demonstrate clearly that our various extensions to the ADR model are perfectly "backward-compatible" with them.

5 Reasoning in the absence of call/cc

In this section, we examine some reasoning principles that are *enabled* by removing call/cc from our language.

Consider this variant of the “awkward” example (from ADR):

$$\begin{aligned}\tau &= (\mathsf{unit} \to \mathsf{unit}) \to \mathsf{int}\\ e_1 &= \mathsf{let}\ x = \mathsf{ref}\ 0\ \mathsf{in}\\ &\qquad \lambda f.\,(x := 0; f\ \langle\rangle; x := 1; f\ \langle\rangle; !x)\\ e_2 &= \lambda f.\,(f\ \langle\rangle; f\ \langle\rangle; 1)\end{aligned}$$

What has changed is that now the callback is run twice, and in e_1, the first call to f is preceded by the assignment of x to 0, not 1.

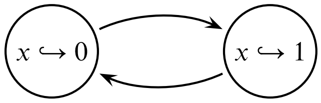
It is easy to see that e_1 and e_2 are not equivalent in **HOSC** (or even **FOSC**). In particular, here is a distinguishing context C:

$$\begin{aligned}&\mathsf{let}\ g = \bullet\ \mathsf{in}\ \mathsf{let}\ b = \mathsf{ref}\ \mathsf{ff}\ \mathsf{in}\\ &\mathsf{let}\ f = (\lambda\_.\ \mathsf{if}\ !b\ \mathsf{then}\ \mathsf{call/cc}\,(k.\ g\ (\lambda\_.\,\mathsf{throw}\ \langle\rangle\ \mathsf{to}\ k))\\ &\qquad\qquad\qquad \mathsf{else}\ b := \mathsf{tt})\ \mathsf{in}\\ &g\ f\end{aligned}$$

Exploiting its ability to capture the continuation k of the second call to f, the context C is able to set x back to 0 and then immediately throw control back to k. It is easy to verify that $C[e_1]$ yields 0, while $C[e_2]$ yields 1.

In the absence of call/cc, however, computations are “well-bracketed.” Here, this means that whenever x is set to 0, it will eventually be set to 1—no matter what the callback function does. Consequently, it seems intuitively clear that these programs are equivalent in **HOS** (and **FOS**), but how do we prove it? The STS model we have developed so far will clearly not do the job, precisely because that model is *compatible* with call/cc and this example is not. So the question remains: how can we augment the power of our STSs so that they take advantage of well-bracketing?

To see how to answer this question, let us see what goes wrong if we try to give an STS for our well-bracketed equivalence. First, recall the STS (from Section 3) that we used in order to prove the original awkward example. To see why this STS is insufficient for our present purposes, suppose the function value resulting from evaluating e_1—call it v_1—is applied in the $x \hookrightarrow 1$ state.[2] The first thing that happens is that x is set to 0. However, as there is no transition from the $x \hookrightarrow 1$ state to the $x \hookrightarrow 0$ state, there is no way we can continue the proof. So how about adding that transition?



While adding the transition from $x \hookrightarrow 1$ to $x \hookrightarrow 0$ clears the first hurdle, it also erects a new one: according to the STS, it is now possible that, after the second call to f, we end up in the left state—even though this situation (x pointing to 0 after that call) cannot actually arise in reality. And indeed, if x could point to 0 at that point, our

[2] When proving functions logically related, we must consider the possibility that they are invoked in an arbitrary “future” world—i.e., a world where our STS may be in any state that is reachable from its initial state. This ensures *monotonicity* of the logical relation (Theorem 1, Section 7.1).

proof would be doomed. In summary, while we would like to add this transition, we also want to keep the context from using it. This is where *private transitions* come in.

5.1 Private transitions

Private transitions are a new class of transitions in our state transition systems, separate from the ordinary transitions that we have seen so far (and which we henceforth call *public transitions*). The basic idea is very simple: when reasoning about the relatedness of terms, we must show that—when viewed *extensionally*—they appear only to be making public transitions, and correspondingly we may assume that the context only makes public transitions as well. Internally, however, *within* a computation, we may make use of both public and private transitions.

Concretely, we can use the following STS to prove our running example[3] (where the dashed arrow denotes a private transition):

$$\begin{tikzcd}
\boxed{x \hookrightarrow 0} \arrow[r, bend left] & \boxed{x \hookrightarrow 1} \arrow[l, bend left, dashed]
\end{tikzcd}$$

First, if v_1 is called in the starting state $x \hookrightarrow 1$, the presence of the private transition allows us to "lawfully" transition from $x \hookrightarrow 1$ to $x \hookrightarrow 0$. Second, we know that, because we are in the $x \hookrightarrow 1$ state before the second call to f and there is no public transition from there to any other state, we must still be in that same state when f returns. Hence, we know that x points to 1 at that point, as desired. Lastly, although the body of v_1 makes a private transition internally (when called in starting state $x \hookrightarrow 1$), it appears extensionally to make a public transition, since its final state ($x \hookrightarrow 1$) is obviously publicly accessible from whichever state was the initial one.

Private transitions let us prove not only this example, but also several others from the literature that hold exclusively in the absence of call/cc [including Pitts and Stark's "higher-order profiling" example (Pitts & Stark, 1998)—see Section 9 for details]. The intuitive reason why private transitions "don't work" with call/cc is that, in the presence of call/cc, every time we pass control to the context may be the last. Therefore, the requirement that the extensional behavior of a term must appear like a public transition would essentially imply that every internal transition must be public as well.

The "callback with lock" example. Here is another equivalence (from ADR) that holds in **HOS** but not in **HOSC**. Interestingly, this example was provable in the original ADR model, but only through some complex step-index hackery. The proof we are about to sketch is much cleaner and easier to understand.

Consider the following two encodings of a counter object with two methods: an *increment* function that also takes a callback argument, which it invokes, and a *poll*

[3] A detailed formal proof of this equivalence is given in Section 9.

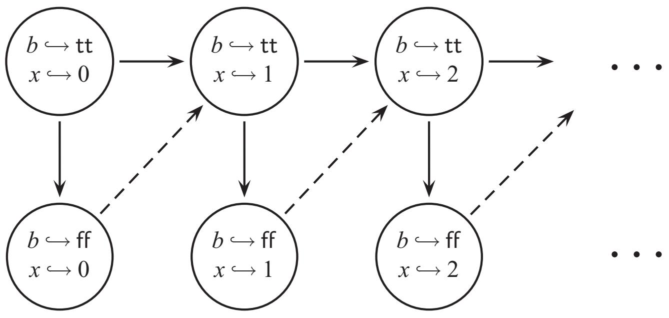

$b \hookrightarrow \mathsf{tt}$ $x \hookrightarrow 0$
$b \hookrightarrow \mathsf{tt}$ $x \hookrightarrow 1$
$b \hookrightarrow \mathsf{tt}$ $x \hookrightarrow 2$
$b \hookrightarrow \mathsf{ff}$ $x \hookrightarrow 0$
$b \hookrightarrow \mathsf{ff}$ $x \hookrightarrow 1$
$b \hookrightarrow \mathsf{ff}$ $x \hookrightarrow 2$
. . .
. . .

Fig. 2. STS for the "callback with lock" example in **HOS**.

function that returns the current counter value.

$$\begin{aligned}
\tau &= ((\mathsf{unit} \to \mathsf{unit}) \to \mathsf{unit}) \times (\mathsf{unit} \to \mathsf{int})\\
e_1 &= C[f\ \langle\rangle\,; x := !x + 1]\\
e_2 &= C[\mathsf{let}\ n = !x\ \mathsf{in}\ f\ \langle\rangle\,; x := n + 1]\\
C &= \mathsf{let}\ b = \mathsf{ref}\ \mathsf{tt}\ \mathsf{in}\ \mathsf{let}\ x = \mathsf{ref}\ 0\ \mathsf{in}\\
&\qquad \langle \lambda f.\, \mathsf{if}\ !b\ \mathsf{then}\ b := \mathsf{ff}\,; \bullet\,; b := \mathsf{tt}\ \mathsf{else}\ \langle\rangle,\\
&\qquad\ \lambda\_.\ !x\rangle
\end{aligned}$$

Note that in the second program the counter x is dereferenced *before* the callback is executed, and in the first program it is dereferenced *after*. In both programs, a Boolean lock b guards the increment of the counter, thereby enforcing that running the callback will not result in any change to the counter.

It is not hard to construct a context that exploits the combination of call/cc and higher-order state in order to distinguish e_1 and e_2. The basic idea is to pass the increment method a callback that captures its current continuation and stores that in a ref cell so it can be invoked later. The definition of this distinguishing context appears in Section 9.1.

In the absence of call/cc, however, the two programs are equivalent. To prove this, we employ the infinite STS shown in Figure 2.

For each number n there are two states: one (the "unlocked" state) saying that b points to tt and x points to n in both programs, and another (the "locked" state) saying that b points to ff and x points to n in both programs. It is thus easy to see that the two poll methods are related (they return the same number). To show the increment methods related, suppose they are executed in a state where x points to some m and b points to tt (the other case where $b \hookrightarrow \mathsf{ff}$ is trivial). Before invoking the callback, b is set to ff and, in the second program, n is bound to m. Accordingly, we move "downwards" in our STS to the locked state and can then call f. Because that state does not have any other public successors, we will still be there if and when f returns—indeed, this is the essence of what it means to be a "locked" state. In the first program, x is then incremented, i.e., set to $m + 1$. In the second program, x is set to $n+1 = m+1$. Finally, b is set back to tt and we thus move to the matching private successor ($b \hookrightarrow \mathsf{tt}$, $x \hookrightarrow m + 1$) in the STS. Since this is a public successor of the initial state ($b \hookrightarrow \mathsf{tt}$, $x \hookrightarrow m$), our extensional transition appears public and we are done.

It is worth noting that the STS in Figure 2 is actually more "precise" than necessary for proving the desired program equivalence. In particular, for the purpose of proving e_1 and e_2 equivalent, it would suffice to collapse all states in which $b \hookrightarrow \mathsf{tt}$ holds down to a single state, in which case there would exist private transitions from every state to every other state but (as before) no public transitions going from any $b \hookrightarrow \mathsf{ff}$ states to the lone $b \hookrightarrow \mathsf{tt}$ state. However, the added precision of the STS in Figure 2 would prove critically useful if one were to extend the objects defined by e_1 and e_2 with a third method, *testmono*, testing the *monotonicity* of state change in e_1 and e_2—i.e., the way that the integer pointed to by x only increases over time. For example, e_1 might define *testmono*, of type $\mathsf{unit} \to \mathsf{unit} \to \mathsf{bool}$, as

$$\lambda\_.\, \mathsf{let}\ y = !x\ \mathsf{in}\ \lambda\_.\, !x \geqslant y$$

and e_2 might define *testmono* as

$$\lambda\_.\, \lambda\_.\, \mathsf{tt}$$

Starting in a state where $x \hookrightarrow n$ holds, these two implementations of *testmono* are only logically related if we know that, in all (privately or publicly accessible) future states, x will point to an integer no less than n. The STS in Figure 2 guarantees this property by omitting any transitions (public or private) from $x \hookrightarrow n$ to $x \hookrightarrow m$ states, where $m < n$.

5.2 *Inconsistent states*

While private transitions are clearly a useful extension to our STS model, there is one kind of "well-bracketed example" we are aware of that private transitions alone are insufficient to account for. We are referring to the "deferred divergence" example, presented by ADR as an example they could not handle. The original version of this equivalence, due to O'Hearn & Reddy (1995), was presented in the setting of Idealized Algol, and it does not hold in the presence of higher-order state. (We will consider a variant of O'Hearn's example later on, in Section 6.) Here, we consider a version of the equivalence that *does* hold in **HOS**, based on the one in Bohr's thesis (Bohr, 2007):

$$\begin{aligned}
\tau &= ((\mathsf{unit} \to \mathsf{unit}) \to \mathsf{unit}) \to \mathsf{unit}\\
e_1 &= \mathsf{let}\ x = \mathsf{ref}\ \mathsf{ff}\ \mathsf{in}\ \mathsf{let}\ y = \mathsf{ref}\ \mathsf{ff}\ \mathsf{in}\\
&\quad\ \lambda f.\, f\ (\lambda\_.\, \mathsf{if}\ !x\ \mathsf{then}\ \bot\ \mathsf{else}\ y := \mathsf{tt});\\
&\qquad\quad \mathsf{if}\ !y\ \mathsf{then}\ \bot\ \mathsf{else}\ x := \mathsf{tt}\\
e_2 &= \lambda f.\, f\ (\lambda\_.\, \bot)
\end{aligned}$$

Intuitively, the explanation why e_1 and e_2 are equivalent goes as follows. The functions returned by both programs take a higher-order callback f as an argument and apply it to a thunk. In the case of e_2, if that thunk argument ($\lambda\_.\, \bot$, where $\bot$ is a divergent term) is ever applied, either during the call to f or at some point in the future (e.g., if the thunk were stored by f in a ref cell and then called later), then the program will clearly diverge. Now, e_1 implements the same divergence behavior, but in a rather sneaky way. It maintains two private flags x and y, initially set to ff. If the thunk that it passes to f is applied *during* the call to f, then the thunk's body will not immediately diverge (as in the case of e_2), but rather merely set y to tt. Then, if and when f returns, e_1 will check if y points to tt and, if so, diverge. If the thunk was not applied during the call to f, then e_1 will set x to tt, thus ensuring that any future attempt to apply the thunk will diverge as well.

As in the previous examples, note that this equivalence does not hold in the presence of call/cc. Here is a distinguishing context:

$$\mathsf{call/cc}\,(k.\ \bullet\ (\lambda g.\,\mathsf{throw}\ g\ \langle\rangle\ \mathsf{to}\ k))$$

To prove the equivalence in **HOS**, we can split the proof into two directions of approximation. Proving that e_2 approximates e_1 is actually very easy because (1) it is trivial to show that $\lambda\_.\,\bot$ approximates the thunk that e_1 passes to f, and (2) if a program $C[e_2]$ terminates (which is the assumption of observational approximation), then $C[e_1]$ must in fact maintain the invariant that $y \hookrightarrow \mathsf{ff}$, and using that invariant the proof is totally straightforward.

In contrast, the other direction of approximation seems at first glance impossible to prove using logical relations. The issue is that we have to show that the thunks passed to the callback f are related, i.e., that $\lambda\_.\,\mathsf{if}\ !x\ \mathsf{then}\ \bot\ \mathsf{else}\ y := \mathsf{tt}$ approximates $\lambda\_.\,\bot$, which obviously is *false* since, when applied (as they may be) in a state where x points to ff, the first converges while the second diverges.

To solve this conundrum, we do the blindingly obvious thing, which is to introduce *falsehood* into our model. Specifically, we extend our STSs with *inconsistent states*, in which we can prove false things, such as that a terminating computation approximates a divergent one. How, one may ask, can this possibly work? The idea is as follows: when we enter an inconsistent state, we effectively shift the proof burden from the logical relation for terms to the logical relation for *continuations*. That is, while it becomes very easy to prove that two terms are related in an inconsistent state, it becomes very *hard* to prove that two continuations K_1 and K_2 are related in such a state—in most cases, we will be forced to prove that K_1 diverges. Thus, while inconsistent states do allow a limited kind of falsehood inside an approximation proof, we can only enter into them if we *know* that the continuation of the term on the left-hand side of the approximation will diverge anyway.

Concretely, to show that e_1 approximates e_2, we construct the STS given in Figure 3, where the diamond indicates an inconsistent state: For the moment, ignore the top-left state (we explain it below). In the proof, we wish to show that the thunks passed to the callback f are logically related in the top-right state, which requires showing that they are related in any state accessible from it. Fortunately, this is easy. If the thunks are called in the bottom-left state, then they both diverge. If they are called in the top-right or bottom-right state, then the else-branch is executed (in the first program) and we move to (or stay in) the bottom-right state—since this state is inconsistent, the proof is trivially done.

Dually, we must show that the continuations of the callback applications are also related in any state (publicly) accessible from the top-right one. If the continuations

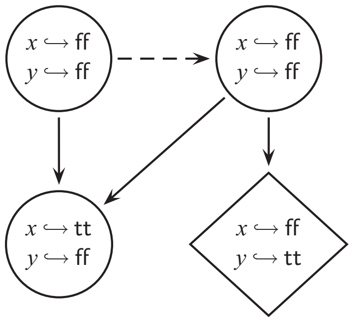


Fig. 3. STS for a variant of the "deferred divergence" example.

are invoked in the top-right or the bottom-left state, they will set x to tt, thereby transitioning to the bottom left. If, on the other hand, they are invoked in the inconsistent bottom-right state, then we are required to show that the first one diverges, which fortunately it will since y points to tt.

Now about the top-left state, whose heap constraint is identical to the one in the top-right state: the reason for including this state has to do with soundness of the logical relation. In order to ensure soundness, we require that when an STS is installed in the possible world, it may not contain any inconsistent states that are *publicly* accessible from its starting state. We say in this case that the starting state is *safe*. (Without this safety restriction, it would be easy to show, for instance, that tt approximates ff in any world W by simply adding an STS to W with a single inconsistent state.)

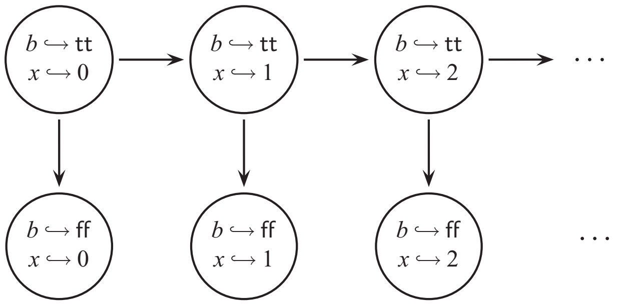
To circumvent this restriction, we use the top-left state as our starting state and connect it to the top-right state by a private transition. (In the proof, the first step before invoking the callbacks is to transition into the top-right state.) This is fine so long as the extensional behavior of the functions we are relating makes a public transition, and here it does—if they are invoked in the top-left state, then either they diverge or they return control in the bottom-left state, which is publicly accessible from the top left.

6 Reasoning with first-order state

In this section, we consider an orthogonal restriction to the one examined in the previous section. Instead of removing call/cc from the language, what happens if we restrict state to be first order? What new reasoning principles are enabled by this restriction?

6.1 Backtracking

Recall the "callback with lock" example from Section 5.1, which we proved equivalent in **HOS**. As it turns out, that equivalence also holds in **FOSC**. Of course, we would not be able to prove that using the **HOSC** model since the equivalence does not hold in **HOSC**. But let us see what exactly goes wrong if we try. First of all,



$$\begin{tikzcd}
\boxed{\begin{matrix} b \hookrightarrow \mathsf{tt} \\ x \hookrightarrow 0 \end{matrix}} \arrow[r] \arrow[d] & \boxed{\begin{matrix} b \hookrightarrow \mathsf{tt} \\ x \hookrightarrow 1 \end{matrix}} \arrow[r] \arrow[d] & \boxed{\begin{matrix} b \hookrightarrow \mathsf{tt} \\ x \hookrightarrow 2 \end{matrix}} \arrow[r] \arrow[d] & \cdots \\
\boxed{\begin{matrix} b \hookrightarrow \mathsf{ff} \\ x \hookrightarrow 0 \end{matrix}} & \boxed{\begin{matrix} b \hookrightarrow \mathsf{ff} \\ x \hookrightarrow 1 \end{matrix}} & \boxed{\begin{matrix} b \hookrightarrow \mathsf{ff} \\ x \hookrightarrow 2 \end{matrix}} & \cdots
\end{tikzcd}$$

Fig. 4. STS for the “callback with lock” example in **FOSC**.

recall the use of private transitions in our earlier proof. Due to call/cc, we cannot use any private transitions this time. Clearly, making them public is not an option, so what if we just drop them entirely? In the resulting STS, shown in Figure 4, we still know that running the callback in a locked state ($b \hookrightarrow \mathsf{ff}$, $x \hookrightarrow m$) will leave us in the very same state if and when it returns. However, without any outgoing (private) transition from that state, it seems that we are subsequently stuck.

Fortunately, we are not. The insight now is that the absence of higher-order state allows us to do *backtracking* within our STS. Concretely, we can backtrack from the locked state to the unlocked state we were in before ($b \hookrightarrow \mathsf{tt}$, $x \hookrightarrow m$), and then transition (publicly) to its successor ($b \hookrightarrow \mathsf{tt}$, $x \hookrightarrow m+1$). Intuitively, this kind of backtracking would not be sound in the presence of higher-order state because, in that setting, the callback might have stored some higher-order data during its execution (such as functions or continuations) that are only logically related in the locked state and its successors.[4] Since ($b \hookrightarrow \mathsf{tt}$, $x \hookrightarrow m+1$) is not a successor of the previous locked state, the final heaps would then fail to satisfy the final world in which the increment functions return. Here in the first-order setting, though, there is no way for the callback to store such higher-order data, so backtracking is perfectly sound.

We use the term “backtracking” purposefully (if informally) to suggest that, while this technique allows more flexibility in proofs, it does not permit transitioning to arbitrary states in the STS. Rather, when we prove that two functions are logically related, we must show they behave the same when applied in any *starting state* s, which could be any state of the STS. Backtracking means that, inside the proof that the functions are related, we can transition from any state accessible from s to any other state accessible from s (by first backtracking to s and then making a normal transition), but not to any states inaccessible from s. For instance, in the backtracking proof sketch for callback-with-lock above, it was fine to transition from ($b \hookrightarrow \mathsf{ff}$, $x \hookrightarrow m$) to ($b \hookrightarrow \mathsf{tt}$, $x \hookrightarrow m+1$) because both of them were accessible from the state ($b \hookrightarrow \mathsf{tt}$, $x \hookrightarrow m$) in which the proof began, but it would not have been okay to transition to ($b \hookrightarrow \mathsf{tt}$, $x \hookrightarrow m-1$) for instance.

[4] Indeed, the context that distinguishes between the two programs in **HOSC** employs precisely such a callback, namely one that stores its current continuation in a ref cell. It is given in Section 9.1.

To see why we only permit transitions to states that are reachable from the starting state s, consider how transitions are used within a proof. When we transition to a state s', it means either that we are about to pass control to related callbacks in that state—e.g., the transition to $(b \hookrightarrow \mathsf{ff}, x \hookrightarrow m)$ above—or that we have finished executing the computation of the function bodies and are ready to return related values in that state—e.g., the transition to $(b \hookrightarrow \mathsf{tt}, x \hookrightarrow m+1)$ above. In the first case, it is necessary for s' to be accessible from s because the callbacks are parameters of the functions being related and, as such, are only known to be logically related in states accessible from the state s in which the functions were invoked. In the second case, s' must be publicly accessible from s because the end-to-end behavior of the functions is required to follow a public transition.

A precise technical explanation of how the model is changed to allow backtracking, and why this is sound, will be given in Section 7.3.

6.2 Putting it together

The example we just looked at might suggest that backtracking is mainly useful as a replacement for private transitions in the presence of call/cc. But in fact, they are complementary techniques. In particular, for equivalences that hold only in **FOS** but not in **HOS** or **FOSC**, we can profitably employ backtracking, private transitions, and inconsistent states, all working together.

Consider this simpler version of the "deferred divergence" example, based closely on an example of O'Hearn & Reddy (1995):

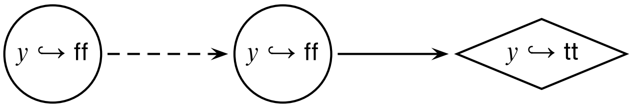
$$\begin{aligned}
\tau &= ((\mathsf{unit} \rightarrow \mathsf{unit}) \rightarrow \mathsf{unit}) \rightarrow \mathsf{unit}\\
e_1 &= \mathsf{let}\ y = \mathsf{ref\ ff\ in}\\
&\quad\ \lambda f.\, f\ (\lambda\_.\, y := \mathsf{tt});\\
&\qquad\quad \mathsf{if}\ !y\ \mathsf{then}\ \bot\ \mathsf{else}\ \langle\rangle\\
e_2 &= \lambda f.\, f\ (\lambda\_.\, \bot)
\end{aligned}$$

These programs are not only distinguishable in the setting of **FOSC** (by the same distinguishing context as given in Section 5.2), but also in **HOS**, as the following context C demonstrates:

$$\mathsf{let}\ r = \mathsf{ref}\ (\lambda\_.\, \langle\rangle)\ \mathsf{in}\ \bullet\ (\lambda g.\, r := g);\, !r\ \langle\rangle$$

It is easy to verify that $C[e_1]$ terminates, while $C[e_2]$ diverges.

The two programs are, however, equivalent in **FOS**, which we can prove using the following STS:



The proof is largely similar to (if a bit simpler than) the one sketched for the higher-order version of this example in Section 5.2. We start in the left state and transition immediately along the private transition to the middle state. With the help of the inconsistent right state, it is easy to show that the thunk arguments passed to the callback are related in the middle state. Hence, when the callback returns, we are either in the right state or the middle state. In the former case, we must show that the continuation in the left-hand-side program diverges; in the latter, we *backtrack* to the initial, left state, which is of course publicly accessible from itself. (We will present this proof in more detail below, in Section 9.2.)

Why, one might ask, is it not possible to avoid the use of backtracking here by adding a private transition back from the middle state to the left state? (Of course, it *must* not be possible, or else the equivalence would hold true in **HOS**, which as we have seen it does not.) The answer is that, if we were to add such a transition, then we would not be able to prove that the thunk arguments to the callback f were logically related in the middle state. Specifically, in order to show the latter, we must show that the thunks are related in any state accessible (by any kind of transition) from the middle state. So if there were any transition from the middle to the left state, we would have to show that the thunks were related starting in the left state as well—but they are not, because there is no public transition from the initial left state to the inconsistent right state, and adding one would be unsound.

7 Technical development

We now present the models for our various languages formally. It is easiest to start with the model for **HOS**, and then show how small changes to that yield the models for **HOSC**, **FOS**, and **FOSC**.

As described in Section 3, we employ a step-indexed Kripke logical relation, which is a kind of possible-worlds model.

Worlds. Figure 5 displays the construction of worlds, along with various related operations and relations.[5] Worlds W consist of a step index k, heap typings Σ_1 and Σ_2 (for the first and second programs, respectively), and an array of islands $\omega = \iota_1, \ldots, \iota_m$. (We sometimes write $W(i)$ to refer to ι_i.) Islands in turn are (possibly infinite) state transition systems governing disjoint pieces of the heap. Each consists of a current state s, a transition relation δ, a public transition relation φ, a set of inconsistent states $\lightning$, and last but not least, a mapping H from states to heap constraints (in the form of world-indexed heap relations—more on that below). The public transition relation φ must be a subset of the "full" transition relation δ (note: the private transitions are obtained by subtracting φ from δ), and we require both δ and φ to be reflexive and transitive.

What exactly "states" s are—i.e., how we define the state space State—does not really matter. That is, State is essentially a parameter of the model, except that it needs to be at least large enough to encode bijections on memory locations (see our relational interpretation of `ref` types below). For our purposes, we find it convenient to assume that State contains all terms and all sets of terms. Also, note that while an island's H map is defined on *all* states in State, we typically only care about how

[5] Here and in the following development, we use the dot-notation to project components out of a structure. As an example, we write $W.\Sigma_1$ to extract the first heap typing out of a world W.

$$\begin{array}{rcl}
\mathrm{HeapAtom}_n & \stackrel{\mathrm{def}}{=} & \{(W,h_1,h_2) \mid W \in \mathrm{World}_n\} \\
\mathrm{HeapRel}_n & \stackrel{\mathrm{def}}{=} & \{\psi \subseteq \mathrm{HeapAtom}_n \mid \forall (W,h_1,h_2) \in \psi.\ \forall W' \sqsupseteq W.\ (W',h_1,h_2) \in \psi\} \\
\mathrm{Island}_n & \stackrel{\mathrm{def}}{=} & \{\iota = (s,\delta,\varphi,\lightning,H) \mid s \in \mathrm{State} \wedge \delta \subseteq \mathrm{State}^2 \wedge \varphi \subseteq \delta \wedge \delta,\varphi \text{ reflexive} \wedge \\
 & & \quad \delta,\varphi \text{ transitive} \wedge \lightning \subseteq \mathrm{State} \wedge H \in \mathrm{State} \to \mathrm{HeapRel}_n\} \\
\mathrm{World}_n & \stackrel{\mathrm{def}}{=} & \{W = (k,\Sigma_1,\Sigma_2,\omega) \mid k < n \wedge \exists m.\ \omega \in \mathrm{Island}_k^m\} \\
\mathrm{ContAtom}_n[\tau_1,\tau_2] & \stackrel{\mathrm{def}}{=} & \{(W,K_1,K_2) \mid W \in \mathrm{World}_n \wedge W.\Sigma_1;\cdot;\cdot \vdash K_1 \div \tau_1 \wedge W.\Sigma_2;\cdot;\cdot \vdash K_2 \div \tau_2\} \\
\mathrm{TermAtom}_n[\tau_1,\tau_2] & \stackrel{\mathrm{def}}{=} & \{(W,e_1,e_2) \mid W \in \mathrm{World}_n \wedge W.\Sigma_1;\cdot;\cdot \vdash e_1 : \tau_1 \wedge W.\Sigma_2;\cdot;\cdot \vdash e_2 : \tau_2\}
\end{array}$$

$$\begin{array}{rcl}
\mathrm{HeapAtom}[\tau_1,\tau_2] & \stackrel{\mathrm{def}}{=} & \bigcup_n \mathrm{HeapAtom}_n[\tau_1,\tau_2] \\
\mathrm{World} & \stackrel{\mathrm{def}}{=} & \bigcup_n \mathrm{World}_n \\
\mathrm{ContAtom}[\tau_1,\tau_2] & \stackrel{\mathrm{def}}{=} & \bigcup_n \mathrm{ContAtom}_n[\tau_1,\tau_2] \\
\mathrm{TermAtom}[\tau_1,\tau_2] & \stackrel{\mathrm{def}}{=} & \bigcup_n \mathrm{TermAtom}_n[\tau_1,\tau_2]
\end{array}$$

$$\begin{array}{rcl}
\mathrm{ValRel}[\tau_1,\tau_2] & \stackrel{\mathrm{def}}{=} & \{r \subseteq \mathrm{TermAtom}^{\mathsf{val}}[\tau_1,\tau_2] \mid \forall (W,v_1,v_2) \in r.\ \forall W' \sqsupseteq W.\ (W',v_1,v_2) \in r\} \\
\mathrm{SomeValRel} & \stackrel{\mathrm{def}}{=} & \{R = (\tau_1,\tau_2,r) \mid r \in \mathrm{ValRel}[\tau_1,\tau_2]\}
\end{array}$$

$$\begin{array}{rclrcl}
\lfloor(\iota_1,\ldots,\iota_m)\rfloor_k & \stackrel{\mathrm{def}}{=} & (\lfloor\iota_1\rfloor_k,\ldots,\lfloor\iota_m\rfloor_k) & \lfloor H\rfloor_k & \stackrel{\mathrm{def}}{=} & \lambda s.\lfloor H(s)\rfloor_k \\
\lfloor(s,\delta,\varphi,\lightning,H)\rfloor_k & \stackrel{\mathrm{def}}{=} & (s,\delta,\varphi,\lightning,\lfloor H\rfloor_k) & \lfloor\psi\rfloor_k & \stackrel{\mathrm{def}}{=} & \{(W,h_1,h_2) \in r \mid W.k < k\}
\end{array}$$

$$\begin{array}{rcl}
\triangleright(k+1,\Sigma_1,\Sigma_2,\omega) & \stackrel{\mathrm{def}}{=} & (k,\Sigma_1,\Sigma_2,\lfloor\omega\rfloor_k) \\
\triangleright r & \stackrel{\mathrm{def}}{=} & \{(W,e_1,e_2) \mid W.k > 0 \Rightarrow (\triangleright W,e_1,e_2) \in r\}
\end{array}$$

$$\begin{array}{rcrcl}
(k',\Sigma_1',\Sigma_2',\omega') & \sqsupseteq & (k,\Sigma_1,\Sigma_2,\omega) & \stackrel{\mathrm{def}}{=} & k' \leq k \wedge \Sigma_1' \supseteq \Sigma_1 \wedge \Sigma_2' \supseteq \Sigma_2 \wedge \omega' \sqsupseteq \lfloor\omega\rfloor_{k'} \\
(\iota_1',\ldots,\iota_{m'}') & \sqsupseteq & (\iota_1,\ldots,\iota_m) & \stackrel{\mathrm{def}}{=} & m' \geq m \wedge \forall j \in \{1,\ldots,m\}.\ \iota_j' \sqsupseteq \iota_j \\
(s',\delta',\varphi',\lightning',H') & \sqsupseteq & (s,\delta,\varphi,\lightning,H) & \stackrel{\mathrm{def}}{=} & (\delta',\varphi',\lightning',H') = (\delta,\varphi,\lightning,H) \wedge (s,s') \in \delta \\
(k',\Sigma_1',\Sigma_2',\omega') & \sqsupseteq^{\mathsf{pub}} & (k,\Sigma_1,\Sigma_2,\omega) & \stackrel{\mathrm{def}}{=} & k' \leq k \wedge \Sigma_1' \supseteq \Sigma_1 \wedge \Sigma_2' \supseteq \Sigma_2 \wedge \omega' \sqsupseteq^{\mathsf{pub}} \lfloor\omega\rfloor_{k'} \\
(\iota_1',\ldots,\iota_{m'}') & \sqsupseteq^{\mathsf{pub}} & (\iota_1,\ldots,\iota_m) & \stackrel{\mathrm{def}}{=} & m' \geq m \wedge \forall j \in \{1,\ldots,m\}.\ \iota_j' \sqsupseteq^{\mathsf{pub}} \iota_j \wedge \\
 & & & & \forall j \in \{m+1,\ldots,m'\}.\ \mathrm{safe}(\iota_j') \\
(s',\delta',\varphi',\lightning',H') & \sqsupseteq^{\mathsf{pub}} & (s,\delta,\varphi,\lightning,H) & \stackrel{\mathrm{def}}{=} & (\delta',\varphi',\lightning',H') = (\delta,\varphi,\lightning,H) \wedge (s,s') \in \varphi
\end{array}$$

$$\mathrm{safe}(W) \stackrel{\mathrm{def}}{=} \forall \iota \in W.\omega.\ \mathrm{safe}(\iota) \qquad \mathrm{safe}(\iota) \stackrel{\mathrm{def}}{=} \forall s'.\ (\iota.s,s') \in \iota.\varphi \Rightarrow s' \notin \iota.\lightning$$

$$\mathrm{consistent}(W) \stackrel{\mathrm{def}}{=} \nexists \iota \in W.\omega.\ \iota.s \in \iota.\lightning$$

$$\psi \otimes \psi' \stackrel{\mathrm{def}}{=} \{(W,h_1 \uplus h_1',h_2 \uplus h_2') \mid (W,h_1,h_2) \in \psi \wedge (W,h_1',h_2') \in \psi'\}$$

$$(h_1,h_2) : W \stackrel{\mathrm{def}}{=} \vdash h_1 : W.\Sigma_1 \wedge \vdash h_2 : W.\Sigma_2 \wedge (W.k > 0 \Rightarrow (\triangleright W,h_1,h_2) \in \bigotimes\{\iota.H(\iota.s) \mid \iota \in W.\omega\})$$

Fig. 5. Worlds and auxiliary definitions.

it is defined on a particular set of "states of interest"—whether there is other junk in the State space is irrelevant.

Based on the two transition relations (full and public), we define two notions of future worlds (aka world extension). First, we say that W' *extends* W, written $W' \sqsupseteq W$, iff it contains the same islands as W (and possibly more), and for each island in W, the new state s' of that island in W'—which is the only aspect of the island that is permitted to change in future worlds—is accessible from the old state s in W, according to the island's full transition relation δ. *Public extension*, written $W' \sqsupseteq^{\mathsf{pub}} W$, is defined analogously, except using the public transition relation φ instead of δ, and with the additional requirement that the new islands (those in W' but not in W) must be *safe*. An island is safe iff there is no public transition from its current state to any inconsistent state.

The reason why our (and ADR's) heap relations are world-indexed is that, when expressing heap constraints, we want to be able to say, for instance, that a value in the first heap must be logically related to a value in the second heap. In that case, we need to have some way of talking about the "current" world under which that logical relation should be considered, and by world-indexing the heap relations we enable the current world to be passed in as a parameter. These world-indexed heap relations are quite restricted, however. Specifically, they must be monotone with respect to world extension, meaning that heaps related in one world will continue to be related in any future world. This ensures that adding a new island to the world, or making (any kind of) transition within an existing island, does not violate the heap constraints of other islands.

The last two definitions also concern heap relations. Two heaps h_1 and h_2 *satisfy a world* W, written $(h_1, h_2) : W$, iff they can be split into disjoint subheaps such that for each island in W there is a subheap of h_1 and a corresponding subheap of h_2 that are related by that island's current heap relation (the relation associated with the island's current state). A heap relation ψ is the *tensor* of ψ' and ψ'', written $\psi' \otimes \psi''$, if it contains all (W, h_1, h_2) that can be split into disjoint parts $(W, h_1', h_2') \in \psi'$ and $(W, h_1'', h_2'') \in \psi''$.

7.1 HOS

Logical relation. Our logical relation for **HOS** is defined in Figure 6. The value relation $\mathrm{V}[\![\tau]\!]\rho$ is fairly standard. The environment ρ gives meaning to the free type variables of τ: for each such variable, ρ stores a semantic (i.e., relational) interpretation and, for the sake of enforcing well-typedness, two syntactic interpretations (one for the first program and one for the second). We write ρ_1 and ρ_2 for the first and second syntactic substitutions obtained from ρ.

The only real difference between our value relation and the one from the ADR model is in $\mathrm{V}[\![\mathsf{ref}\,\tau]\!]\rho$, the interpretation of reference types. Basically, we say that two references l_1 and l_2 are logically related at type $\mathsf{ref}\,\tau$ in world W if there exists an island ι in W (we write $W(i)$ to mean the ith island in W), such that (1) ι's heap constraint (in any reachable state) requires of l_1 and l_2 precisely that their contents are related at type τ, and (2) the reachable states in ι encode a bijection between locations that includes the pair (l_1, l_2). The latter condition is needed in order to model the presence of reference equality testing $l_1 == l_2$ in the language. It employs an auxiliary "bij" function that can for instance be defined as follows (assuming State contains sets of language values):

$$\mathsf{bij}(s) \stackrel{\text{def}}{=} \begin{cases} \{(l_1, l_2) \mid \langle l_1, l_2 \rangle \in s\} & \text{if that is a partial bijection} \\ \emptyset & \text{otherwise} \end{cases}$$

$$\begin{array}{rcl}
\mathrm{V}\llbracket\alpha\rrbracket\rho & \overset{\text{def}}{=} & \rho(\alpha).r \\
\mathrm{V}\llbracket b\rrbracket\rho & \overset{\text{def}}{=} & \{(W,v,v)\in\mathrm{TermAtom}[b,b]\} \\
\mathrm{V}\llbracket\tau\times\tau'\rrbracket\rho & \overset{\text{def}}{=} & \{(W,\langle v_1,v_1'\rangle,\langle v_2,v_2'\rangle)\in\mathrm{TermAtom}[\rho_1(\tau\times\tau'),\rho_2(\tau\times\tau')]\mid \\
& & \quad (W,v_1,v_2)\in\mathrm{V}\llbracket\tau\rrbracket\rho\wedge(W,v_1',v_2')\in\mathrm{V}\llbracket\tau'\rrbracket\rho\} \\
\mathrm{V}\llbracket\tau'\to\tau\rrbracket\rho & \overset{\text{def}}{=} & \{(W,\lambda x{:}\tau_1.\,e_1,\lambda x{:}\tau_2.\,e_2)\in\mathrm{TermAtom}[\rho_1(\tau'\to\tau),\rho_2(\tau'\to\tau)]\mid \\
& & \quad \forall W',v_1,v_2.\ W'\sqsupseteq W\wedge(W',v_1,v_2)\in\mathrm{V}\llbracket\tau'\rrbracket\rho\Rightarrow \\
& & \quad (W',e_1[v_1/x],e_2[v_2/x])\in\mathrm{E}\llbracket\tau\rrbracket\rho\} \\
\mathrm{V}\llbracket\forall\alpha.\,\tau\rrbracket\rho & \overset{\text{def}}{=} & \{(W,\Lambda\alpha.e_1,\Lambda\alpha.e_2)\in\mathrm{TermAtom}[\rho_1(\forall\alpha.\,\tau),\rho_2(\forall\alpha.\,\tau)]\mid \\
& & \quad \forall W'\sqsupseteq W.\ \forall(\tau_1,\tau_2,r)\in\mathrm{SomeValRel}. \\
& & \quad (W',e_1[\tau_1/\alpha],e_2[\tau_2/\alpha])\in\mathrm{E}\llbracket\tau\rrbracket\rho,\alpha\mapsto(\tau_1,\tau_2,r)\} \\
\mathrm{V}\llbracket\exists\alpha.\,\tau\rrbracket\rho & \overset{\text{def}}{=} & \{(W,\mathsf{pack}\ \langle\tau_1,v_1\rangle\ \mathsf{as}\ \tau_1',\mathsf{pack}\ \langle\tau_2,v_2\rangle\ \mathsf{as}\ \tau_2')\in\mathrm{TermAtom}[\rho_1(\exists\alpha.\,\tau),\rho_2(\exists\alpha.\,\tau)]\mid \\
& & \quad \exists r.\ (\tau_1,\tau_2,r)\in\mathrm{SomeValRel}\wedge(W,v_1,v_2)\in\mathrm{V}\llbracket\tau\rrbracket\rho,\alpha\mapsto(\tau_1,\tau_2,r)\} \\
\mathrm{V}\llbracket\mu\alpha.\,\tau\rrbracket\rho & \overset{\text{def}}{=} & \{(W,\mathsf{roll}_{\tau_1}\ v_1,\mathsf{roll}_{\tau_2}\ v_2)\in\mathrm{TermAtom}[\rho_1(\mu\alpha.\,\tau),\rho_2(\mu\alpha.\,\tau)]\mid \\
& & \quad (W,v_1,v_2)\in\triangleright\mathrm{V}\llbracket\tau[\mu\alpha.\,\tau/\alpha]\rrbracket\rho\} \\
\mathrm{V}\llbracket\mathsf{ref}\ \tau\rrbracket\rho & \overset{\text{def}}{=} & \{(W,l_1,l_2)\in\mathrm{TermAtom}[\rho_1(\mathsf{ref}\ \tau),\rho_2(\mathsf{ref}\ \tau)]\mid\exists i.\ \forall W'\sqsupseteq W. \\
& & \quad (l_1,l_2)\in\mathsf{bij}(W'(i).s)\wedge\exists\psi.\ W'(i).H(W'(i).s)= \\
& & \quad \psi\otimes\{(\widetilde{W},\{l_1\mapsto v_1\},\{l_2\mapsto v_2\})\in\mathrm{HeapAtom}\mid(\widetilde{W},v_1,v_2)\in\mathrm{V}\llbracket\tau\rrbracket\rho\}\}
\end{array}$$

$$\mathrm{O}\ \overset{\text{def}}{=}\ \{(W,e_1,e_2)\mid\forall h_1,h_2.\ (h_1,h_2):W\wedge\langle h_1;e_1\rangle\downarrow^{<W.k}\Rightarrow\mathrm{consistent}(W)\wedge\langle h_2;e_2\rangle\downarrow\}$$

$$\begin{array}{rcl}
\mathrm{K}\llbracket\tau\rrbracket\rho & \overset{\text{def}}{=} & \{(W,K_1,K_2)\in\mathrm{ContAtom}[\rho_1(\tau),\rho_2(\tau)]\mid \\
& & \quad \forall W',v_1,v_2.\ W'\sqsupseteq^{\mathsf{pub}}W\wedge(W',v_1,v_2)\in\mathrm{V}\llbracket\tau\rrbracket\rho\Rightarrow(W',K_1[v_1],K_2[v_2])\in\mathrm{O}\}
\end{array}$$

$$\begin{array}{rcl}
\mathrm{E}\llbracket\tau\rrbracket\rho & \overset{\text{def}}{=} & \{(W,e_1,e_2)\in\mathrm{TermAtom}[\rho_1(\tau),\rho_2(\tau)]\mid \\
& & \quad \forall K_1,K_2.\ (W,K_1,K_2)\in\mathrm{K}\llbracket\tau\rrbracket\rho\Rightarrow(W,K_1[e_1],K_2[e_2])\in\mathrm{O}\}
\end{array}$$

$$\begin{array}{rcl}
\mathrm{G}\llbracket\cdot\rrbracket\rho & \overset{\text{def}}{=} & \{(W,\emptyset)\mid W\in\mathrm{World}\} \\
\mathrm{G}\llbracket\Gamma,x{:}\tau\rrbracket\rho & \overset{\text{def}}{=} & \{(W,(\gamma,x\mapsto(v_1,v_2)))\mid(W,\gamma)\in\mathrm{G}\llbracket\Gamma\rrbracket\rho\wedge(W,v_1,v_2)\in\mathrm{V}\llbracket\tau\rrbracket\rho\} \\
\mathrm{D}\llbracket\cdot\rrbracket & \overset{\text{def}}{=} & \{\emptyset\} \\
\mathrm{D}\llbracket\Delta,\alpha\rrbracket & \overset{\text{def}}{=} & \{\rho,\alpha\mapsto R\mid\rho\in\mathrm{D}\llbracket\Delta\rrbracket\wedge R\in\mathrm{SomeValRel}\} \\
\mathrm{S}\llbracket\cdot\rrbracket & \overset{\text{def}}{=} & \mathrm{World} \\
\mathrm{S}\llbracket\Sigma,l{:}\tau\rrbracket & \overset{\text{def}}{=} & \mathrm{S}\llbracket\Sigma\rrbracket\cap\{W\in\mathrm{World}\mid(W,l,l)\in\mathrm{V}\llbracket\mathsf{ref}\ \tau\rrbracket\emptyset\}
\end{array}$$

$$\begin{array}{rcl}
\Sigma;\Delta;\Gamma\vdash e_1\lesssim_{\log}e_2:\tau & \overset{\text{def}}{=} & \Sigma;\Delta;\Gamma\vdash e_1:\tau\wedge\Sigma;\Delta;\Gamma\vdash e_2:\tau\wedge \\
& & \forall W,\rho,\gamma.\ W\in\mathrm{S}\llbracket\Sigma\rrbracket\wedge\rho\in\mathrm{D}\llbracket\Delta\rrbracket\wedge(W,\gamma)\in\mathrm{G}\llbracket\Gamma\rrbracket\rho\Rightarrow \\
& & (W,\rho_1\gamma_1e_1,\rho_2\gamma_2e_2)\in\mathrm{E}\llbracket\tau\rrbracket\rho
\end{array}$$

Fig. 6. A step-indexed biorthogonal Kripke logical relation for **HOS**.

Our formulation of $\mathrm{V}\llbracket\mathsf{ref}\ \tau\rrbracket\rho$ is slightly different from ADR's and a bit more flexible—e.g., ours can be used to prove Bohr's "local state release" example (Bohr, 2007) (see the technical appendix, Dreyer *et al.*, 2012), whereas ADR's cannot—but this added flexibility does not affect any of our "headlining" examples from Sections 3–6.

As explained in Section 4, the value relation is lifted to a term relation via biorthogonality. Concretely, we define the continuation relation $\mathrm{K}\llbracket\tau\rrbracket\rho$ based on $\mathrm{V}\llbracket\tau\rrbracket\rho$, and then the term relation $\mathrm{E}\llbracket\tau\rrbracket\rho$ based on $\mathrm{K}\llbracket\tau\rrbracket\rho$:

- Two continuations are related iff they yield related observations when applied to related values.
- Two terms are related iff they yield related observations when evaluated under related continuations.

Yielding related observations here means (see the definition of O) that, whenever two heaps satisfy the world W in question and the first program terminates in the first heap (within $W.k$ steps), then the second program terminates in the second heap and the world is *consistent* (i.e., no island is in an inconsistent state). This corresponds to the intuition given in Section 5.2 that an inconsistent world is one in which the first program diverges.

Notice that the continuation relation quantifies only over *public* future worlds. This captures the essential idea (explained in Section 5.1) that the context can only make public transitions. In order to see this, it is important to understand how a typical proof in a biorthogonal logical relation goes. Roughly, showing the relatedness of two programs that involve a call to an unknown function (e.g., a callback) eventually reduces to showing that the continuations of the function call are related; thanks to the definition of $\mathrm{K}[\![\tau]\!]\rho$, we will only need to consider the possibility that those continuations are invoked in a *public* future world of the world we were in prior to the function call—in other words, we can assume that the function call made a public transition. We will see how this works in detail in the example proofs in Section 9.

Finally, the logical relation is lifted to open terms in the usual way, quantifying over related closing substitutions ρ and γ matching Δ and Γ, respectively, as well as an initial world in which every location bound in Σ is related to itself. We write γ_1 and γ_2 here as shorthand for the first and second value substitutions contained in γ.

Step indexing Our use of step indexing to stratify the construction of worlds and to define the logical relation follows the development in ADR quite closely. In order to avoid a circularity, the various universes of discourse are defined by induction on natural numbers (top of Figure 5). Note that World_n is defined in terms of Island_k for $k < n$. We write World to mean the limit $\bigcup_n \mathrm{World}_n$, and similarly for some other semantic classes.

When comparing two generations of an island in the definition of our world extensions, the $\lfloor\cdot\rfloor_k$ operator is used to cut down the heap relations of the old one to the level of the new one. Using the "later" operator $\triangleright$, world satisfaction (bottom of Figure 5) requires each pair of subheaps to be related not right away but one step later. Intuitively, this is safe because it takes a step of computation to dereference a pointer.

The logical relation itself is also defined by induction on natural numbers (in addition to the usual induction on types). In particular, the value relation at a recursive type, $\mathrm{V}[\![\mu\alpha.\,\tau]\!]\rho$, refers to the value relation at a potentially larger type but uses $\triangleright$ to decrease its step index. For further details about step indexing we refer the interested reader to the literature (Ahmed *et al.*, 2009; Dreyer *et al.*, 2010).

Basic properties We highlight some of the many basic properties that are used all the time in logical relations proofs.

Frequently, we assume that we are given some related values (e.g., as inputs to functions), and we want them to be still related after we have added an island to the world or made a transition. It is therefore crucial that, like heap relations, value relations are monotone with respect to world extension. Since we enforce this property for relational interpretations of abstract types (see the definition of ValRel in Figure 5), it is easy to show that the value relation indeed has this property:

Lemma 1 (*Monotonicity of the value relation*)
If $W' \sqsupseteq W$ and $(W, v_1, v_2) \in \mathrm{V}[\![\tau]\!]\rho$, then $(W', v_1, v_2) \in \mathrm{V}[\![\tau]\!]\rho$.

Another important property of the value relation is that it is included in the term relation. Consequently, when showing two values related by the term relation, it suffices to show them related by the value relation.

Lemma 2 (*Term relation includes value relation*)
$\mathrm{V}[\![\tau]\!]\rho \subseteq \mathrm{E}[\![\tau]\!]\rho$

Equally useful in proofs is the following lemma, which recognizes the monadic binding that occurs implicitly in ML-like languages (Dreyer *et al.*, 2010, 2011). It applies when reasoning about two programs that contain related terms e_1 and e_2 in evaluation position. In that case, it is okay to forget about e_1 and e_2, and replace them with unknown related values in an unknown public future world.

Lemma 3 (*Monadic bind*)
If $(W, e_1, e_2) \in \mathrm{E}[\![\tau]\!]\rho$
and $\forall W' \sqsupseteq^{\mathsf{pub}} W.\ \forall v_1, v_2.\ (W', v_1, v_2) \in \mathrm{V}[\![\tau]\!]\rho \Rightarrow (W', K_1[v_1], K_2[v_2]) \in \mathrm{E}[\![\tau']\!]\rho$,
then $(W, K_1[e_1], K_2[e_2]) \in \mathrm{E}[\![\tau']\!]\rho$.

Proof

- Suppose $(W, K_1', K_2') \in \mathrm{K}[\![\tau']\!]\rho$.
- We must show $(W, K_1'[K_1[e_1]], K_2'[K_2[e_2]]) \in \mathrm{O}$.
- By the first premise this reduces to showing $(W, K_1'[K_1], K_2'[K_2]) \in \mathrm{K}[\![\tau]\!]\rho$.
- So suppose $W' \sqsupseteq^{\mathsf{pub}} W$ and $(W', v_1, v_2) \in \mathrm{V}[\![\tau]\!]\rho$.
- To show: $(W', K_1'[K_1[v_1]], K_2'[K_2[v_2]]) \in \mathrm{O}$
- From the second premise we know $(W', K_1[v_1], K_2[v_2]) \in \mathrm{E}[\![\tau']\!]\rho$.
- The claim follows then from $(W, K_1', K_2') \in \mathrm{K}[\![\tau']\!]\rho$ and monotonicity.

$\square$

The following lemma essentially states that the observation relation is closed under pure expansion (not involving heap mutation) on either side.

Lemma 4 (*Closure under pure expansion*)
If $\langle h; e_1\rangle \downarrow^{<W.k}$ implies $\langle h; e_1'\rangle \downarrow^{<W.k}$ and $\langle h; e_2'\rangle \downarrow$ implies $\langle h; e_2\rangle \downarrow$ for any h, then $(W, e_1', e_2') \in \mathrm{O}$ implies $(W, e_1, e_2) \in \mathrm{O}$.

Finally, the following lemma collects some facts about syntactic well-typedness of logical relation components. The judgments it mentions governing well-formedness of value and type substitutions are defined in the standard way (see Dreyer *et al.*, 2012).

Lemma 5 (*Syntactic typing properties*)

1. If $W \in \mathrm{S}[\![\Sigma]\!]$, then $\Sigma \sqsubseteq W.\Sigma_1$ and $\Sigma \sqsubseteq W.\Sigma_2$.
2. If $\rho \in \mathrm{D}[\![\Delta]\!]$, then $\cdot \vdash \rho_1 : \Delta$ and $\cdot \vdash \rho_2 : \Delta$.
3. If $(W, \gamma) \in \mathrm{G}[\![\Gamma]\!]\rho$, then $W.\Sigma_1; \cdot; \cdot \vdash \gamma_1 : \rho_1\Gamma$ and $W.\Sigma_2; \cdot; \cdot \vdash \gamma_2 : \rho_2\Gamma$.

Soundness and completeness The proof that our logical relation is sound with respect to **HOS**'s contextual approximation follows closely that of ADR (Ahmed *et al.*, 2009). The basic building blocks are the *compatibility* lemmas (Pitts, 2005), which state that the logical relation is closed under each of the language's constructs; we omit them here as they are fairly standard (they can be found in Dreyer *et al.*, 2012, though). Together, they yield the *fundamental property* of the logical relation (Theorem 1) and the fact that the logical relation is a precongruence with respect to language contexts C (Lemma 7).

Theorem 1 (*Fundamental property*)
If $\Sigma; \Delta; \Gamma \vdash e : \tau$, then $\Sigma; \Delta; \Gamma \vdash e \lesssim_{\log} e : \tau$.

Proof
By induction on the typing derivation, in each case using the appropriate compatibility lemma. □

Lemma 6 (*Weakening*)
If $\Sigma; \Delta; \Gamma \vdash e_1 \lesssim_{\log} e_2 : \tau$ and $\Sigma \sqsubseteq \Sigma'$, $\Delta \subseteq \Delta'$, $\Gamma \subseteq \Gamma'$, $\Delta' \vdash \Gamma'$, then $\Sigma'; \Delta'; \Gamma' \vdash e_1 \lesssim_{\log} e_2 : \tau$.

Lemma 7 (*Precongruence*)
If $\Sigma; \Delta; \Gamma \vdash e_1 \lesssim_{\log} e_2 : \tau$ and $\vdash C : (\Sigma; \Delta; \Gamma; \tau) \rightsquigarrow (\Sigma'; \Delta'; \Gamma'; \tau')$, then $\Sigma'; \Delta'; \Gamma' \vdash C[e_1] \lesssim_{\log} C[e_2] : \tau'$.

Proof
By induction on the derivation of the context typing, in each case using the corresponding compatibility lemma. For a context containing subterms we also need Theorem 1. The rule for an empty context is

$$\frac{\Sigma \sqsubseteq \Sigma' \quad \Delta \subseteq \Delta' \quad \Gamma \subseteq \Gamma'}{\vdash \bullet : (\Sigma; \Delta; \Gamma; \tau) \rightsquigarrow (\Sigma'; \Delta'; \Gamma'; \tau)}$$

and hence that case requires Lemma 6. □

It remains to prove *adequacy* of the logical relation (Pitts, 2005), i.e., the fact that if two closed terms are logically related and the first one terminates under a (well-formed) heap, then so does the second (under the same heap). This involves the construction of a canonical *safe* world for a given heap typing. Details of the construction can be found in Dreyer *et al.* (2012).

Lemma 8 (*Canonical world*)
If $\vdash h : \Sigma$, then for any k there is $W \in \mathrm{S}[\![\Sigma]\!]$ such that $W.k = k$ and $(h, h) : W$ and $\mathrm{safe}(W)$.

Lemma 9 (Adequacy)
If $\Sigma;\cdot;\cdot \vdash e_1 \lesssim_{\log} e_2 : \tau$ and $\vdash h : \Sigma$ and $\langle h; e_1\rangle\downarrow$, then $\langle h; e_2\rangle\downarrow$.

Proof

- Say $\langle h; e_1\rangle\downarrow^j$.
- By Lemma 8 there is $W \in \mathrm{S}[\![\Sigma]\!]$ with $W.k = j+1$, $(h,h) : W$, and safe(W).
- Instantiating $\Sigma;\cdot;\cdot \vdash e_1 \lesssim_{\log} e_2 : \tau$ yields $(W, e_1, e_2) \in \mathrm{E}[\![\tau]\!]$.
- It thus suffices to show $(W, \bullet, \bullet) \in \mathrm{K}[\![\tau]\!]$.
- So suppose $W' \sqsupseteq^{\mathsf{pub}} W$, $(W', v_1, v_2) \in \mathrm{V}[\![\tau]\!]$, $(h_1, h_2) : W'$ and $\langle h_1; v_1\rangle\downarrow^{<W'.k}$.
- We need to show consistent(W') and $\langle h_2; v_2\rangle\downarrow$.
- The former follows from safe(W) because public world extension preserves safety, and the latter is immediate.

□

Theorem 2 (Soundness)
$\lesssim_{\log} \subseteq \lesssim_{\mathrm{ctx}}$

Proof

- Suppose $\Sigma;\Delta;\Gamma \vdash e_1 \lesssim_{\log} e_2 : \tau$ as well as $\vdash C : (\Sigma;\Delta;\Gamma;\tau) \rightsquigarrow (\Sigma';\cdot;\cdot;\tau')$, $\vdash h : \Sigma'$, and $\langle h; C[e_1]\rangle\downarrow$.
- By precongruence (Lemma 7), $\Sigma';\cdot;\cdot \vdash C[e_1] \lesssim_{\log} C[e_2] : \tau'$.
- By adequacy (Lemma 9), $\langle h; C[e_2]\rangle\downarrow$.

□

Following Pitts & Stark (1998), we show completeness of our logical relation with respect to contextual approximation with the help of Mason and Talcott's *ciu*-approximation (Mason & Talcott, 1991) as an intermediate relation.

Definition 1 (CIU approximation)

$$\begin{aligned}\Sigma;\Delta;\Gamma \vdash e_1 \lesssim_{\mathrm{ciu}} e_2 : \tau \ \stackrel{\mathrm{def}}{=}\ & \Sigma;\Delta;\Gamma \vdash e_1 : \tau \wedge \Sigma;\Delta;\Gamma \vdash e_2 : \tau \wedge \forall \delta,\gamma,K,\Sigma',h.\\ & \cdot \vdash \delta : \Delta \wedge \Sigma';\cdot;\cdot \vdash \gamma : \delta\Gamma \wedge \Sigma';\cdot;\cdot \vdash K \div \delta\tau \wedge \Sigma \subseteq \Sigma' \wedge\\ & \vdash h : \Sigma' \wedge \langle h; K[\delta\gamma e_1]\rangle\downarrow \Rightarrow \langle h; K[\delta\gamma e_2]\rangle\downarrow\end{aligned}$$

Theorem 3 (Completeness)
$\lesssim_{\mathrm{ctx}} \subseteq \lesssim_{\mathrm{ciu}} \subseteq \lesssim_{\log}$

Proof
Proving the inclusion of $\lesssim_{\mathrm{ctx}}$ in $\lesssim_{\mathrm{ciu}}$ is fairly easy (details can be found in Dreyer *et al.*, 2012). The inclusion of $\lesssim_{\mathrm{ciu}}$ in $\lesssim_{\log}$ follows as an almost immediate consequence of the Fundamental Property, together with the logical relation's biorthogonal definition:

- Suppose $W \in \mathrm{S}[\![\Sigma]\!]$, $\rho \in \mathrm{D}[\![\Delta]\!]$, and $(W,\gamma) \in \mathrm{G}[\![\Gamma]\!]\rho$.
- To show: $(W, \rho_1\gamma_1 e_1, \rho_2\gamma_2 e_2) \in \mathrm{E}[\![\tau]\!]\rho$
- So suppose $(W, K_1, K_2) \in \mathrm{K}[\![\tau]\!]\rho$, $(h_1, h_2) : W$, and $\langle h_1; K_1[\rho_1\gamma_1 e_1]\rangle\downarrow^{<W.k}$.
- To show: consistent(W) and $\langle h_2; K_2[\rho_2\gamma_2 e_2]\rangle\downarrow$
- By Theorem 1 we know $\Sigma;\Delta;\Gamma \vdash e_1 \lesssim_{\log} e_1 : \tau$.

- Instantiating this yields consistent(W) and $\langle h_2; K_2[\rho_2\gamma_2 e_1]\rangle\downarrow$.
- Using the assumption that $\Sigma;\Delta;\Gamma \vdash e_1 \precsim_{\text{ciu}} e_2 : \tau$, it suffices to show:
  1. $\cdot \vdash \rho_2 : \Delta$ and $W.\Sigma_2;\cdot;\cdot \vdash \gamma_2 : \rho_2\Gamma$, which follow by Lemma 5;
  2. $W.\Sigma_2;\cdot;\cdot \vdash K_2 \div \rho_2\tau$, which holds by definition of ContAtom;
  3. $\Sigma \subseteq W.\Sigma_2$, which follows by Lemma 5; and
  4. $\vdash h_2 : W.\Sigma_2$, which holds by definition of HeapAtom.

□

7.2 HOSC

The model for **HOSC** can be obtained from the one for **HOS** by making two changes. First of all, in **HOSC**, we have to account for the presence of first-class continuation values $\mathsf{cont}_\tau\, K$, i.e., we have to define what it means for two values to be related at type $\mathsf{cont}\,\tau$. We do that by essentially just copying the definition of our continuation relation $\mathrm{K}[\![\tau]\!]\rho$:[6]

$$\begin{aligned}\mathrm{V}[\![\mathsf{cont}\,\tau]\!]\rho \overset{\text{def}}{=} \{(W, \mathsf{cont}\,K_1, \mathsf{cont}\,K_2) \in \mathrm{TermAtom}[\rho_1(\mathsf{cont}\,\tau), \rho_2(\mathsf{cont}\,\tau)] \mid \forall W', v_1, v_2.\\ W' \sqsupseteq W \wedge (W', v_1, v_2) \in \mathrm{V}[\![\tau]\!]\rho \Rightarrow (W', K_1[v_1], K_2[v_2]) \in \mathrm{O}\}\end{aligned}$$

Notice, however, that the definition given here quantifies over arbitrary future worlds rather than just public future worlds. The reason is that the logical relation on values needs to be monotone with respect to $\sqsupseteq$ (Lemma 1), not just $\sqsupseteq^{\mathsf{pub}}$. This (forced) change has serious consequences. Since in **HOSC** any continuation can be injected into the value language, we will therefore need to prove that continuations related by $\mathrm{K}[\![\tau]\!]\rho$ become values related by $\mathrm{V}[\![\mathsf{cont}\,\tau]\!]\rho$ (see Lemma 10). Hence, the strengthening of $\mathrm{V}[\![\mathsf{cont}\,\tau]\!]\rho$ to quantify over arbitrary future worlds forces the corresponding strengthening of $\mathrm{K}[\![\tau]\!]\rho$ as well.

Of course, what this means is that in the presence of call/cc, the private and public transition relations must be collapsed into one, and consequently we must disallow inconsistent states, too. This corresponds to the intuition we gave in Section 5.1, namely that private transitions and inconsistent states are only sound to use in the absence of call/cc. Formally, we can easily implement these restrictions by redefining Island_n as follows:

$$\mathrm{Island}'_n \overset{\text{def}}{=} \{\iota \in \mathrm{Island}_n \mid \iota.\varphi = \iota.\delta \wedge \iota.\lightning = \emptyset\}$$

Under this definition, the two notions of world extension coincide and all worlds are consistent. The rest of the model stays the same. In particular, proofs done in the **HOS** model that do not make use of private transitions or inconsistent states can be transferred without any change. The soundness and completeness proofs carry over as well. The former merely needs to be extended in a straightforward way to deal with cont, $\mathsf{call/cc}$, and throw, as we show below.

It is worth noting that, even in the model for **HOS**, where private transitions and inconsistent states are supported, one may not require the use of both these features

[6] The reason why we do not simply define $\mathrm{V}[\![\mathsf{cont}\,\tau]\!]\rho$ directly in terms of $\mathrm{K}[\![\tau]\!]\rho$ will become apparent in Section 8.

in every equivalence proof. For instance, in Sections 9.1 and 9.4, our proofs employ private transitions but not inconsistent states. In such cases, it is fine to assume the model is restricted to one where all states are consistent, thereby avoiding any need to reason explicitly about consistency of worlds within logical-relations proofs. This is perfectly sound for the same reason that the above restriction of the **HOSC** model (i.e., Island'_n) is—namely, because the proof of soundness and completeness of the logical relation with respect to contextual equivalence does not itself make use of either private transitions and inconsistent states. (The proof of compatibility for reference allocation does rely on the ability to extend the world with a new island, but this only requires public world extension.)

In order to state compatibility for cont we first need to lift $\mathrm{K}[\![\tau]\!]\rho$ to open continuations the same way we lifted $\mathrm{E}[\![\tau]\!]\rho$ to open expressions:

Definition 2

$$\begin{aligned}\Sigma;\Delta;\Gamma \vdash K_1 \precsim_{\log} K_2 \div \tau \;\stackrel{\text{def}}{=}\; & \Sigma;\Delta;\Gamma \vdash K_1 \div \tau \wedge \Sigma;\Delta;\Gamma \vdash K_2 \div \tau \wedge \forall W,\rho,\gamma.\\ & W \in \mathrm{S}[\![\Sigma]\!] \wedge \rho \in \mathrm{D}[\![\Delta]\!] \wedge (W,\gamma) \in \mathrm{G}[\![\Gamma]\!]\rho \Rightarrow\\ & (W,\rho_1\gamma_1K_1,\rho_2\gamma_2K_2) \in \mathrm{K}[\![\tau]\!]\rho\end{aligned}$$

Lemma 10 (*Compatibility: Cont*)

$$\frac{\Sigma;\Delta;\Gamma \vdash K_1 \precsim_{\log} K_2 \div \tau}{\Sigma;\Delta;\Gamma \vdash \mathsf{cont}_\tau\, K_1 \precsim_{\log} \mathsf{cont}_\tau\, K_2 : \mathsf{cont}\,\tau}$$

Proof

- Suppose $W \in \mathrm{S}[\![\Sigma]\!]$, $\rho \in \mathrm{D}[\![\Delta]\!]$, and $(W,\gamma) \in \mathrm{G}[\![\Gamma]\!]\rho$.
- To show: $(W, \mathsf{cont}\ \rho_1\gamma_1K_1, \mathsf{cont}\ \rho_2\gamma_2K_2) \in \mathrm{E}[\![\mathsf{cont}\,\tau]\!]\rho$
- By Lemma 2 it suffices to show $(W, \mathsf{cont}\ \rho_1\gamma_1K_1, \mathsf{cont}\ \rho_2\gamma_2K_2) \in \mathrm{V}[\![\mathsf{cont}\,\tau]\!]\rho$.
- This is implied by $(W,\rho_1\gamma_1K_1,\rho_2\gamma_2K_2) \in \mathrm{K}[\![\tau]\!]\rho$, which we get from the premise.

□

Lemma 11 (*Compatibility: Call/cc*)

$$\frac{\Sigma;\Delta;\Gamma, x{:}\mathsf{cont}\,\tau \vdash e_1 \precsim_{\log} e_2 : \tau}{\Sigma;\Delta;\Gamma \vdash \mathsf{call/cc}_\tau(x.\ e_1) \precsim_{\log} \mathsf{call/cc}_\tau(x.\ e_2) : \tau}$$

Proof

- Suppose $W \in \mathrm{S}[\![\Sigma]\!]$, $\rho \in \mathrm{D}[\![\Delta]\!]$, $(W,\gamma) \in \mathrm{G}[\![\Gamma]\!]\rho$, and $(W,K_1,K_2) \in \mathrm{K}[\![\tau]\!]\rho$.
- To show: $(W, K_1[\mathsf{call/cc}\,(x.\ \rho_1\gamma_1e_1)], K_2[\mathsf{call/cc}\,(x.\ \rho_2\gamma_2e_2)]) \in \mathrm{O}$
- By Lemma 4 it suffices to show $(W, K_1[(\rho_1\gamma_1e_1)[\mathsf{cont}\ K_1/x]], K_2[(\rho_2\gamma_2e_2)[\mathsf{cont}\ K_2/x]]) \in \mathrm{O}$.
- Since $(W,K_1,K_2) \in \mathrm{K}[\![\tau]\!]\rho$ implies $(W,\mathsf{cont}\ K_1,\mathsf{cont}\ K_2) \in \mathrm{V}[\![\mathsf{cont}\,\tau]\!]\rho$, we get $(W,\gamma') \in \mathrm{G}[\![\Gamma, x{:}\mathsf{cont}\,\tau]\!]\rho$ for $\gamma' = \gamma, x\mapsto(\mathsf{cont}\ K_1, \mathsf{cont}\ K_2)$.
- Instantiating the premise now yields $(W,\rho_1\gamma'_1e_1,\rho_2\gamma'_2e_2) \in \mathrm{E}[\![\tau]\!]\rho$.
- Hence $(W, K_1[\rho_1\gamma'_1e_1], K_2[\rho_2\gamma'_2e_2]) \in \mathrm{O}$.

- By definition of ContAtom, K_1 and K_2 contain no free type variables. Consequently, $\rho_1\gamma'_1 e_1 = (\rho_1\gamma_1 e_1)[K_1/x]$ and $\rho_2\gamma'_2 e_2 = (\rho_2\gamma_2 e_2)[K_2/x]$.

□

Lemma 12 (Compatibility: Throw)

$$\frac{\Sigma;\Delta;\Gamma \vdash e_1 \lesssim_{\log} e_2 : \tau' \qquad \Sigma;\Delta;\Gamma \vdash e_3 \lesssim_{\log} e_4 : \mathsf{cont}\,\tau'}{\Sigma;\Delta;\Gamma \vdash \mathsf{throw}_\tau\ e_1\ \mathsf{to}\ e_3 \lesssim_{\log} \mathsf{throw}_\tau\ e_2\ \mathsf{to}\ e_4 : \tau}$$

Proof

- Suppose $W \in \mathrm{S}[\![\Sigma]\!]$, $\rho \in \mathrm{D}[\![\Delta]\!]$, $(W,\gamma) \in \mathrm{G}[\![\Gamma]\!]\rho$.
- To show: $(W, \mathsf{throw}\ \rho_1\gamma_1 e_1\ \mathsf{to}\ \rho_1\gamma_1 e_3, \mathsf{throw}\ \rho_2\gamma_2 e_2\ \mathsf{to}\ \rho_2\gamma_2 e_4) \in \mathrm{E}[\![\tau]\!]\rho$
- By instantiating the premises and applying Lemma 3 twice, it suffices to show $(W', \mathsf{throw}\ v_1\ \mathsf{to}\ (\mathsf{cont}\ K_3), \mathsf{throw}\ v_2\ \mathsf{to}\ (\mathsf{cont}\ K_4)) \in \mathrm{E}[\![\tau]\!]\rho$, where
  1. $W' \sqsupseteq^{\mathsf{pub}} W$
  2. $(W', v_1, v_2) \in \mathrm{V}[\![\tau']\!]\rho$
  3. $(W', \mathsf{cont}\ K_3, \mathsf{cont}\ K_4) \in \mathrm{V}[\![\mathsf{cont}\,\tau']\!]\rho$
- So suppose $(W', K_1, K_2) \in \mathrm{K}[\![\tau]\!]\rho$.
- To show: $(W', K_1[\mathsf{throw}\ v_1\ \mathsf{to}\ (\mathsf{cont}\ K_3)], K_2[\mathsf{throw}\ v_2\ \mathsf{to}\ (\mathsf{cont}\ K_4)]) \in \mathrm{O}$
- Note that by Lemma 4 it suffices to show $(W', K_3[v_1], K_4[v_2]) \in \mathrm{O}$.
- Hence instantiating (3) with (2) yields the claim.

□

7.3 FOS

In the first-order state setting, observe that, for the types of values that can be stored in the heap—namely, those of base type—our logical relation for values coincides with syntactic equality. Consequently, when expressing that two heap values are logically related, we no longer need to refer to a world. Obtaining the model for **FOS** from the one for **HOS** is therefore very simple—all that is needed is to remove the ability of heap relations to be world-dependent:

$$\mathrm{HeapRel}'_n \stackrel{\text{def}}{=} \mathscr{P}(\mathrm{Heap} \times \mathrm{Heap})$$

Our heap relations are now more or less the same as in Pitts & Stark (1998)—that is, they are simply heap relations. Correspondingly, we must also update the definitions of $(h_1, h_2) : W$, $\psi' \otimes \psi''$, and $\mathrm{V}[\![\mathsf{ref}\,\tau]\!]\rho$, all in the obvious manner, to reflect the lack of world indices in heap relations. (For details, see Dreyer *et al.*, 2012.) Note that while step-indices are no longer needed to stratify our worlds, they are still useful in modeling general recursive types.

This simplification of HeapRel enables backtracking (see Section 6.1) by isolating islands from one another completely. Specifically, in the model of **HOS**, changing the state of an island ι in a way not prescribed by ι's transition system could have the effect of breaking the heap constraints in *other* islands, since those heap constraints were allowed to depend on the state of the world as a whole and were only required to be monotone under advancement to *future* worlds. In contrast, with the simplified

FOS model, there is now no way for any change to ι's state to affect the satisfaction of other islands' heap constraints, since those heap constraints are not permitted to depend in any way on the state of the world as a whole. This gives us the freedom to backtrack within ι's transition system as we like.

7.4 FOSC

The changes to the **HOS** model discussed in Sections 7.2 and 7.3 are completely orthogonal and may be easily combined in order to obtain a fully abstract model for **FOSC**.

8 Reasoning in the presence of exceptions

In this paper, we have focused attention on first-class continuations as our control effect of interest, and demonstrated that their absence enables the extension of our STS-based Kripke model with the mechanisms of private transitions and inconsistent states. It is natural, then, to ask about the impact that other control effects have on our model. At least in the case of simple, global *exceptions*, the answer is quite straightforward, as we now briefly explain. We conjecture that this basic story also applies to more sophisticated (generative) exception mechanisms like standard MLs, but we leave the consideration of those—as well as of other control effects such as delimited continuations—to future work.

First of all, unlike throwing to a continuation, raising an exception causes a "well-bracketed" kind of control effect, in the sense that it passes control to the exception handler that was most recently pushed onto the control stack. Thus, the presence of exceptions does not *per se* restrict our STS model: we are free to use STSs with private transitions and inconsistent states.

However, the possibility of exceptional behavior means that, when proving two *continuations* to be logically related (by $\mathrm{K}[\![\tau]\!]\rho$), we must show that they behave in a related manner not only when they are plugged with related values, but also when they are passed related raised exceptions. Concretely, consider the language extension presented in Figure 7. It adds a new base type exn that is inhabited by a fixed set of exception constants c_{exn}, and provides constructs for comparing, raising, and catching them. (We write "K does not try" to mean that K's hole is not surrounded by any exception handler.) In that setting, the definition of $\mathrm{K}[\![\tau]\!]\rho$ becomes the following:

$$\begin{array}{l}\{(W,K_1,K_2) \in \mathrm{ContAtom}[\rho_1(\tau),\rho_2(\tau)] \mid \forall W',v_1,v_2.\ W' \sqsupseteq^{\mathsf{pub}} W \Rightarrow \\ \quad ((W',v_1,v_2) \in \mathrm{V}[\![\tau]\!]\rho \Rightarrow (W',K_1[v_1],K_2[v_2]) \in \mathrm{O}) \wedge \\ \quad ((W',v_1,v_2) \in \mathrm{V}[\![\mathsf{exn}]\!] \Rightarrow (W',K_1[\mathsf{raise}\ v_1],K_2[\mathsf{raise}\ v_2]) \in \mathrm{O})\}\end{array}$$

where the value relation at exception type is simply the identity:

$$\mathrm{V}[\![\mathsf{exn}]\!]\rho \stackrel{\mathrm{def}}{=} \{(W,v,v) \in \mathrm{TermAtom}[\mathsf{exn},\mathsf{exn}]\}$$

In essence, this new definition is equivalent to $\mathrm{K}[\![\mathbf{M}(\tau)]\!]\rho$, where $\mathbf{M}$ is the *exception monad*—i.e., $\mathbf{M}(\tau) \approx \tau + \mathsf{exn}$.

$$\begin{array}{rcl}
\tau & ::= & \ldots \mid \mathsf{exn} \\
e & ::= & \ldots \mid c_{\mathsf{exn}} \mid e_1 = e_2 \mid \mathsf{raise}_\tau\ e \mid \mathsf{try}\ e_1\ \mathsf{with}\ x.e_2 \\
v & ::= & \ldots \mid c_{\mathsf{exn}} \\
K & ::= & \ldots \mid \mathsf{raise}_\tau\ K \mid \mathsf{try}\ K\ \mathsf{with}\ x.e_2
\end{array}$$

$$\begin{array}{lll}
\ldots & & \\
\langle h; K[v = v]\rangle & \hookrightarrow \langle h; K[\mathsf{tt}]\rangle & \\
\langle h; K[v_1 = v_2]\rangle & \hookrightarrow \langle h; K[\mathsf{ff}]\rangle & (v_1 \neq v_2) \\
\langle h; K[\mathsf{raise}_\tau\ v]\rangle & \hookrightarrow \langle h; \mathsf{raise}_\tau\ v\rangle & (K \neq \bullet \text{ and } K \text{ does not try}) \\
\langle h; K_1[\mathsf{try}\ K_2[\mathsf{raise}_\tau\ v]\ \mathsf{with}\ x.e_2]\rangle & \hookrightarrow \langle h; K_1[e_2[v/x]]\rangle & (K_2 \text{ does not try}) \\
\langle h; K_1[\mathsf{try}\ v\ \mathsf{with}\ x.e_2]\rangle & \hookrightarrow \langle h; K_1[v]\rangle &
\end{array}$$

$$\ldots \qquad \frac{}{\Sigma;\Delta;\Gamma \vdash c_{\mathsf{exn}} : \mathsf{exn}} \qquad \frac{\Sigma;\Delta;\Gamma \vdash e_1 : \mathsf{exn} \qquad \Sigma;\Delta;\Gamma \vdash e_2 : \mathsf{exn}}{\Sigma;\Delta;\Gamma \vdash e_1 = e_2 : \mathsf{bool}}$$

$$\frac{\Sigma;\Delta;\Gamma \vdash e : \mathsf{exn}}{\Sigma;\Delta;\Gamma \vdash \mathsf{raise}_\tau\ e : \tau} \qquad \frac{\Sigma;\Delta;\Gamma \vdash e_1 : \tau \quad \Sigma;\Delta;\Gamma, x{:}\mathsf{exn} \vdash e_2 : \tau}{\Sigma;\Delta;\Gamma \vdash \mathsf{try}\ e_1\ \mathsf{with}\ x.e_2 : \tau}$$

Fig. 7. Adding simple exceptions.

Both this language extension and the changes to the logical relation can be applied to any of the previously discussed languages and their models. Thus, they really form another axis and result in four additional languages and corresponding models: **FOSE**, **HOSE**, **FOSEC**, and **HOSEC**. For the latter two, note that we do not change the definition $\mathrm{V}[\![\mathsf{cont}\,\tau]\!]\rho$, i.e., the meaning of continuations as values. Doing so would be sound but unnecessary: the only way to use a continuation value $\mathsf{cont}\ K$ is by feeding it a value of the expected type, not an exception, so we need not place any additional conditions on when two continuation values are logically related. Moreover, doing so would keep us from proving some equivalences: for instance, $\bullet$ and $(\mathsf{try}\ \bullet\ \mathsf{with}\ x.\mathsf{tt})$ are certainly not related continuations but, injected into the value language, they *are* contextually equivalent continuation values.

In each of the previous compatibility proofs and in the proof of the adequacy theorem, we need to make the following extension: whenever we show that some continuations are related by $\mathrm{K}[\![\tau]\!]\rho$, we now—as dictated by its new definition—also have to deal with the case that the continuations are passed exceptions. In each case, this is easy since the code in question does not install any exception handlers.

The interesting additional compatibility lemmas are Lemmas 13 and 14:

Lemma 13 (*Compatibility: Raise*)

$$\frac{\Sigma;\Delta;\Gamma \vdash e_1 \lesssim_{\log} e_2 : \mathsf{exn}}{\Sigma;\Delta;\Gamma \vdash \mathsf{raise}_\tau\ e_1 \lesssim_{\log} \mathsf{raise}_\tau\ e_2 : \tau}$$

Proof

- Suppose $W \in \mathrm{S}[\![\Sigma]\!]$, $\rho \in \mathrm{D}[\![\Delta]\!]$, and $(W, \gamma) \in \mathrm{G}[\![\Gamma]\!]\rho$.
- To show: $(W, \mathsf{raise}\ \rho_1\gamma_1 e_1, \mathsf{raise}\ \rho_2\gamma_2 e_2) \in \mathrm{E}[\![\tau]\!]\rho$
- So suppose $(W, K_1, K_2) \in \mathrm{K}[\![\tau]\!]\rho$.

- To show: $(W, K_1[\mathsf{raise}\ \rho_1\gamma_1 e_1], K_2[\mathsf{raise}\ \rho_2\gamma_2 e_2]) \in \mathrm{O}$
- Using the premise, it suffices to show $(W, K_1[\mathsf{raise}\ \bullet], K_2[\mathsf{raise}\ \bullet]) \in \mathrm{K}[\![\mathsf{exn}]\!]\rho$, which decomposes into two parts.
  1. We suppose $(W', v_1, v_2) \in \mathrm{V}[\![\mathsf{exn}]\!]\rho$ for $W' \sqsupseteq^{\mathsf{pub}} W$ and must show
  $$(W', K_1[\mathsf{raise}\ v_1], K_2[\mathsf{raise}\ v_2]) \in \mathrm{O}.$$
  This follows immediately from instantiating $(W, K_1, K_2) \in \mathrm{K}[\![\tau]\!]\rho$.
  2. We suppose $(W', v_1, v_2) \in \mathrm{V}[\![\mathsf{exn}]\!]\rho$ for $W' \sqsupseteq^{\mathsf{pub}} W$ and must show
  $$(W', K_1[\mathsf{raise}\ (\mathsf{raise}\ v_1)], K_2[\mathsf{raise}\ (\mathsf{raise}\ v_2)]) \in \mathrm{O}.$$
  By Lemma 4 this reduces to showing $(W', K_1[\mathsf{raise}\ v_1], K_2[\mathsf{raise}\ v_2]) \in \mathrm{O}$, which we just did in part (1). □

Lemma 14 (*Compatibility: Try*)

$$\frac{\Sigma;\Delta;\Gamma \vdash e_1 \lesssim_{\log} e_2 : \tau \quad \Sigma;\Delta;\Gamma, x{:}\mathsf{exn} \vdash e_3 \lesssim_{\log} e_4 : \tau}{\Sigma;\Delta;\Gamma \vdash \mathsf{try}\ e_1\ \mathsf{with}\ x.e_3 \lesssim_{\log} \mathsf{try}\ e_2\ \mathsf{with}\ x.e_4 : \tau}$$

Proof

- Suppose $W \in \mathrm{S}[\![\Sigma]\!]$, $\rho \in \mathrm{D}[\![\Delta]\!]$, and $(W, \gamma) \in \mathrm{G}[\![\Gamma]\!]\rho$.
- To show: $(W, \mathsf{try}\ \rho_1\gamma_1 e_1\ \mathsf{with}\ x.\rho_1\gamma_1 e_3, \mathsf{try}\ \rho_2\gamma_2 e_2\ \mathsf{with}\ x.\rho_2\gamma_2 e_4) \in \mathrm{E}[\![\tau]\!]\rho$
- So suppose $(W, K_1, K_2) \in \mathrm{K}[\![\tau]\!]\rho$.
- To show: $(W, K_1[\mathsf{try}\ \rho_1\gamma_1 e_1\ \mathsf{with}\ x.\rho_1\gamma_1 e_3], K_2[\mathsf{try}\ \rho_2\gamma_2 e_2\ \mathsf{with}\ x.\rho_2\gamma_2 e_4]) \in \mathrm{O}$
- Using the first premise, it suffices to show
  $$(W, K_1[\mathsf{try}\ \bullet\ \mathsf{with}\ x.\rho_1\gamma_1 e_3], K_2[\mathsf{try}\ \bullet\ \mathsf{with}\ x.\rho_2\gamma_2 e_4]) \in \mathrm{K}[\![\tau]\!]\rho$$
  which decomposes into two parts.
  1. We suppose $(W', v_1, v_2) \in \mathrm{V}[\![\tau]\!]\rho$ for $W' \sqsupseteq^{\mathsf{pub}} W$ and must show
  $$(W', K_1[\mathsf{try}\ v_1\ \mathsf{with}\ x.\rho_1\gamma_1 e_3], K_2[\mathsf{try}\ v_2\ \mathsf{with}\ x.\rho_2\gamma_2 e_4]) \in \mathrm{O}.$$
  By Lemma 4 it suffices to show $(W', K_1[v_1], K_2[v_2]) \in \mathrm{O}$, which follows from $(W, K_1, K_2) \in \mathrm{K}[\![\tau]\!]\rho$.
  2. We suppose $(W', v_1, v_2) \in \mathrm{V}[\![\mathsf{exn}]\!]\rho$ for $W' \sqsupseteq^{\mathsf{pub}} W$ and must show
  $$(W', K_1[\mathsf{try}\ \mathsf{raise}\ v_1\ \mathsf{with}\ x.\rho_1\gamma_1 e_3], K_2[\mathsf{try}\ \mathsf{raise}\ v_2\ \mathsf{with}\ x.\rho_2\gamma_2 e_4]) \in \mathrm{O}.$$
  By Lemma 4 it suffices to show
  $$(W', K_1[(\rho_1\gamma_1 e_3)[v_1/x]], K_2[(\rho_2\gamma_2 e_4)[v_2/x]]) \in \mathrm{O}.$$
  Using monotonicity it is easy to see that $(W', \gamma') \in \mathrm{G}[\![\Gamma, x{:}\mathsf{exn}]\!]\rho$. The second premise then yields $(W', \rho_1\gamma'_1 e_3, \rho_2\gamma'_2 e_4) \in \mathrm{E}[\![\tau]\!]\rho$. Using $(W, K_1, K_2) \in \mathrm{K}[\![\tau]\!]\rho$ and monotonicity we therefore get
  $$(W', K_1[\rho_1\gamma'_1 e_3], K_2[\rho_2\gamma'_2 e_4]) \in \mathrm{O}$$
  for $\gamma' = \gamma, x \mapsto (v_1, v_2)$. Since v_1 and v_2 contain no free type variables by definition of TermAtom, this is what we needed to prove. □

Each of the various examples we have considered in this paper involves proving equivalence of two higher-order functions that, when called, will manipulate some local state and invoke their (unknown) callback arguments. Thus, for each of the examples, the new, more restrictive definition of $\mathrm{K}[\![\tau]\!]\rho$ requires us to consider the possibility that the callback invocation may raise an exception. Since the higher-order function in each example does not install any exception handler around its callback invocation, any exception raised by that callback invocation will remain uncaught, causing the function to return immediately (raising the same exception).

We therefore need to show that any state in which the callback may raise an exception—i.e., any state that is publicly accessible from the one in which the callback was invoked—is also publicly accessible from the initial state in which the higher-order function was called. For the callback-with-lock example, this is indeed the case, since the only state publicly accessible from the "locked" state (in which the callback is invoked) is itself, which is publicly accessible from the "unlocked" starting state. For the other examples, on the other hand, this criterion is not met; and indeed, in the presence of exceptions, it is not hard to find program contexts that distinguish the higher-order functions in those examples.

9 Detailed proofs of selected examples

9.1 Callback with lock

$$\begin{aligned}
\tau\ &= ((\mathsf{unit} \to \mathsf{unit}) \to \mathsf{unit}) \times (\mathsf{unit} \to \mathsf{int})\\
e_1 &= C[f\ \langle\rangle; x := !x + 1]\\
e_2 &= C[\mathsf{let}\ n = !x\ \mathsf{in}\ f\ \langle\rangle; x := n + 1]\\
C &= \mathsf{let}\ b = \mathsf{ref\ tt\ in}\\
&\quad\ \mathsf{let}\ x = \mathsf{ref}\ 0\ \mathsf{in}\\
&\quad\ \langle \lambda f.\ \mathsf{if}\ !b\ \mathsf{then}\ (b := \mathsf{ff}; \bullet; b := \mathsf{tt})\ \mathsf{else}\ \langle\rangle,\\
&\quad\ \ \lambda\_.\ !x\rangle
\end{aligned}$$

Here is a context C' that is able to distinguish e_1 and e_2 in **HOSC**:

$$\begin{aligned}
&\mathsf{let}\ \langle \mathit{inc}, \mathit{poll}\rangle = \bullet\ \mathsf{in}\\
&\mathsf{call/cc}\,(k_0.\ \mathsf{let}\ r = \mathsf{ref}\ k_0\ \mathsf{in}\\
&\qquad\qquad\quad \mathsf{let}\ b = \mathsf{ref\ tt\ in}\\
&\qquad\qquad\quad \mathsf{let}\ g = (\lambda\_.\, \mathsf{call/cc}\,(k.\ r := k))\ \mathsf{in}\\
&\qquad\qquad\quad \mathsf{let}\ h = (\lambda\_.\, b := \mathsf{ff}; \mathsf{throw}\ \langle\rangle\ \mathsf{to}\ !r)\ \mathsf{in}\\
&\qquad\qquad\quad \mathit{inc}\ g; (\mathsf{if}\ !b\ \mathsf{then}\ \mathit{inc}\ h\ \mathsf{else}\ \langle\rangle); \mathsf{if}\ \mathit{poll}\ \langle\rangle = 2\ \mathsf{then}\ \langle\rangle\ \mathsf{else}\ \bot)
\end{aligned}$$

When the second call to the increment method *inc* executes the callback h, the latter jumps back to the continuation that was stored by g in r during the first call to *inc*. In that continuation, n in e_2 is still bound to 0, while x in e_1 points to 1 now. It is easy to verify that $C'[e_1]$ terminates, but $C'[e_2]$ does not.

In the absence of call/cc but presence of exceptions, i.e., in **HOSE**, the programs are equivalent. We show $\cdot;\cdot;\cdot \vdash e_1 \precsim_{\log} e_2 : \tau$ (the other direction is very similar), which immediately reduces to showing $(W, e_1, e_2) \in \mathrm{E}[\![\tau]\!]$ for $W \in$ World. So we suppose

1. $(W, K_1, K_2) \in \mathrm{K}[\![\tau]\!]$
2. $(h_1, h_2) : W$
3. $\langle h_1 ; K_1[e_1]\rangle \downarrow^{<W.k}$

and must now show $\langle h_2 ; K_2[e_2]\rangle \downarrow$. From (3) we know $\langle \widehat{h_1} ; K_1[v_1]\rangle \downarrow^{<W.k}$, where

- $\widehat{h_1} = h_1 \uplus \{l_1^b \mapsto \mathsf{tt}\} \uplus \{l_1^x \mapsto 0\}$
- l_1^b, l_1^x are fresh and distinct
- $v_1 = \langle v_1^{\text{inc}}, v_1^{\text{poll}}\rangle$
- $v_1^{\text{inc}} = \lambda f.\, \mathsf{if}\ !l_1^b\ \mathsf{then}\ (l_1^b := \mathsf{ff} ; f\ \langle\rangle ; l_1^x := !l_1^x + 1 ; l_1^b := \mathsf{tt})\ \mathsf{else}\ \langle\rangle$
- $v_1^{\text{poll}} = \lambda\_.\, !l_1^x$

Similarly, $\langle h_2 ; K_2[e_2]\rangle \downarrow$ if $\langle \widehat{h_2} ; K_2[v_2]\rangle \downarrow$, where

- $\widehat{h_2} = h_2 \uplus \{l_2^b \mapsto \mathsf{tt}\} \uplus \{l_2^x \mapsto 0\}$
- $v_2 = \langle v_2^{\text{inc}}, v_2^{\text{poll}}\rangle$
- $v_2^{\text{inc}} = \lambda f.\, \mathsf{if}\ !l_2^b\ \mathsf{then}\ (l_2^b := \mathsf{ff} ; \mathsf{let}\ n = !l_2^x\ \mathsf{in}\ f\ \langle\rangle ; l_2^x := n + 1 ; l_2^b := \mathsf{tt})\ \mathsf{else}\ \langle\rangle$
- $v_2^{\text{poll}} = \lambda\_.\, !l_2^x$

We now proceed as follows. We first extend W to $\widehat{W}$ with an island governing l_1^b and l_2^b as well as l_1^x and l_2^x, and then show $(\widehat{W}, v_1, v_2) \in \mathrm{V}[\![((\mathsf{unit} \rightarrow \mathsf{unit}) \rightarrow \mathsf{unit}) \times (\mathsf{unit} \rightarrow \mathsf{int})]\!]$, which, combined with (1), concludes the proof.

The island we add is represented by the STS shown in Figure 2 (starting in the top left state). Concretely, we define:

- $\widehat{W} = (W.k, (W.\Sigma_1, l_1^b{:}\mathsf{bool}, l_1^x{:}\mathsf{int}), (W.\Sigma_2, l_2^b{:}\mathsf{bool}, l_2^x{:}\mathsf{int}), (W.\omega, \iota))$
- $\iota = (\langle \mathsf{tt}, 0\rangle, \delta, \varphi, \emptyset, H)$
- $\delta = (\varphi \uplus \{(\langle \mathsf{ff}, i\rangle, \langle \mathsf{tt}, i+1\rangle) \mid i \in \mathbb{N}\})^*$
- $\varphi = (\{(\langle \mathsf{tt}, i\rangle, \langle \mathsf{tt}, i+1\rangle) \mid i \in \mathbb{N}\} \uplus \{(\langle \mathsf{tt}, i\rangle, \langle \mathsf{ff}, i\rangle) \mid i \in \mathbb{N}\})^*$
- $H(\langle o, i\rangle) = \{(\widetilde{W}, \widetilde{h_1}, \widetilde{h_2}) \in \mathrm{HeapAtom} \mid \widetilde{h_1}(l_1^b) = \widetilde{h_2}(l_2^b) = o \wedge \widetilde{h_1}(l_1^x) = \widetilde{h_2}(l_2^x) = i\}$

Here the superscript "*" in the definitions of δ and φ denotes the reflexive, transitive closure over State. Note that $\widehat{W} \sqsupseteq^{\mathsf{pub}} W$ and, using monotonicity, that $(\widehat{h_1}, \widehat{h_2}) : \widehat{W}$. It thus remains to show $(\widehat{W}, v_1, v_2) \in \mathrm{V}[\![((\mathsf{unit} \rightarrow \mathsf{unit}) \rightarrow \mathsf{unit}) \times (\mathsf{unit} \rightarrow \mathsf{int})]\!]$, which decomposes into two parts.

(A) We must show $(\widehat{W}, v_1^{\text{inc}}, v_2^{\text{inc}}) \in \mathrm{V}[\![(\mathsf{unit} \rightarrow \mathsf{unit}) \rightarrow \mathsf{unit}]\!]$. So we suppose

4. $W' \sqsupseteq \widehat{W}$
5. $(W', \lambda y.\, e_3, \lambda y.\, e_4) \in \mathrm{V}[\![\mathsf{unit} \rightarrow \mathsf{unit}]\!]$
6. $(W', K_1', K_2') \in \mathrm{K}[\![\mathsf{unit}]\!]$
7. $(h_1', h_2') : W'$
8. $\langle h_1' ; K_1'[\mathsf{if}\ !l_1^b\ \mathsf{then}\ (l_1^b := \mathsf{ff} ; (\lambda y.\, e_3)\ \langle\rangle ; l_1^x := !l_1^x + 1 ; l_1^b := \mathsf{tt})\ \mathsf{else}\ \langle\rangle]\rangle \downarrow^{<W'.k}$

and need to show

$$\langle h_2' ; K_2'[\mathsf{if}\ !l_2^b\ \mathsf{then}\ (l_2^b := \mathsf{ff} ; \mathsf{let}\ n = !l_2^x\ \mathsf{in}\ (\lambda y.\, e_4)\ \langle\rangle ; l_2^x := n + 1 ; l_2^b := \mathsf{tt})\ \mathsf{else}\ \langle\rangle]\rangle \downarrow.$$

To do so, we inspect our island's state in W' and distinguish two cases:

(a) It is $\langle \mathsf{ff}, i\rangle$ for some i. Then we know from (7) that $h'_1(l_1^b) = h'_2(l_2^b) = \mathsf{ff}$ and $h'_1(l_1^x) = h'_2(l_2^x) = i$. Consequently, we get $\langle h'_1 ; K'_1[\langle\rangle]\rangle \downarrow^{<W'.k}$ from (8). And similarly, if $\langle h'_2 ; K'_2[\langle\rangle]\rangle \downarrow$ then

$$\langle h'_2 ; K'_2[\mathsf{if}\ !l_2^b\ \mathsf{then}\ (l_2^b := \mathsf{ff} ; \mathsf{let}\ n = !l_2^x\ \mathsf{in}\ (\lambda y.\, e_4)\ \langle\rangle ; l_2^x := n+1;$$
$$l_2^b := \mathsf{tt})\ \mathsf{else}\ \langle\rangle]\rangle \downarrow .$$

Finally, since $(W', \langle\rangle, \langle\rangle) \in \mathrm{V}[\![\mathsf{unit}]\!]$, the claim then follows from (6).

(b) It is $\langle \mathsf{tt}, i\rangle$ for some i. Then we know from (7) that $h'_1(l_1^b) = h'_2(l_2^b) = \mathsf{tt}$ and $h'_1(l_1^x) = h'_2(l_2^x) = i$. Consequently, we get

$$\langle h'_1[l_1^b {\mapsto} \mathsf{ff}] ; K'_1[e_3[\langle\rangle/y] ; l_1^x := !l_1^x + 1 ; l_1^b := \mathsf{tt}]\rangle \downarrow^{<W'.k}$$

from (8). And similarly, if $\langle h'_2[l_2^b {\mapsto} \mathsf{ff}] ; K'_2[e_4[\langle\rangle/y] ; l_2^x := i+1 ; l_2^b := \mathsf{tt}]\rangle \downarrow$ then

$$\langle h'_2 ; K'_2[\mathsf{if}\ !l_2^b\ \mathsf{then}\ (l_2^b := \mathsf{ff} ; \mathsf{let}\ n = !l_2^x\ \mathsf{in}\ (\lambda y.\, e_4)\ \langle\rangle ; l_2^x := n+1;$$
$$l_2^b := \mathsf{tt})\ \mathsf{else}\ \langle\rangle]\rangle \downarrow .$$

Let $W'_{\langle \mathsf{ff}, i\rangle}$ be the world obtained from W' by setting our island's state from $\langle \mathsf{tt}, i\rangle$ to $\langle \mathsf{ff}, i\rangle$, so $W'_{\langle \mathsf{ff}, i\rangle} \sqsupseteq^{\mathsf{pub}} W'$. Using monotonicity, it is easy to see that

$$(h'_1[l_1^b {\mapsto} \mathsf{ff}], h'_2[l_2^b {\mapsto} \mathsf{ff}]) : W'_{\langle \mathsf{ff}, i\rangle}.$$

Since $(W'_{\langle \mathsf{ff}, i\rangle}, \langle\rangle, \langle\rangle) \in \mathrm{V}[\![\mathsf{unit}]\!]$, we get $(W'_{\langle \mathsf{ff}, i\rangle}, e_3[\langle\rangle/y], e_4[\langle\rangle/y]) \in \mathrm{E}[\![\mathsf{unit}]\!]$ from (5). Therefore, it now suffices to show

$$(W'_{\langle \mathsf{ff}, i\rangle}, K'_1[\bullet ; l_1^x := !l_1^x + 1 ; l_1^b := \mathsf{tt}], K'_2[\bullet ; l_2^x := i+1 ; l_2^b := \mathsf{tt}]) \in \mathrm{K}[\![\mathsf{unit}]\!] .$$

To this end, we need to consider the continuations to be filled with either related values or related exceptions. For the former, we suppose

9. $W'' \sqsupseteq^{\mathsf{pub}} W'_{\langle \mathsf{ff}, i\rangle}$
10. $(h''_1, h''_2) : W''$
11. $\langle h''_1 ; K'_1[\langle\rangle ; l_1^x := !l_1^x + 1 ; l_1^b := \mathsf{tt}]\rangle \downarrow^{<W''.k}$

and need to show $\langle h''_2 ; K'_2[\langle\rangle ; l_2^x := i+1 ; l_2^b := \mathsf{tt}]\rangle \downarrow$. Notice that in our island the only public successor of state $\langle \mathsf{ff}, i\rangle$ is $\langle \mathsf{ff}, i\rangle$ itself. Therefore, we know from (9) and (10) that $h''_1(l_1^x) = i$ and thus from (11) that

$$\langle h''_1[l_1^x {\mapsto} (i+1)][l_1^b {\mapsto} \mathsf{tt}] ; K'_1[\langle\rangle]\rangle \downarrow^{<W''.k} .$$

Similarly, if $\langle h''_2[l_2^x {\mapsto} (i+1)][l_2^b {\mapsto} \mathsf{tt}] ; K'_2[\langle\rangle]\rangle \downarrow$ then $\langle h''_2 ; K'_2[\langle\rangle ; l_2^x := i + 1 ; l_2^b := \mathsf{tt}]\rangle \downarrow$. Now, let $W''_{\langle \mathsf{tt}, i+1\rangle}$ be the world obtained from W'' by setting our island's state from $\langle \mathsf{ff}, i\rangle$ to $\langle \mathsf{tt}, i+1\rangle$, so $W''_{\langle \mathsf{tt}, i+1\rangle} \sqsupseteq W''$. Using monotonicity it is then easy to see that $(h''_1[l_1^x {\mapsto} (i+1)][l_1^b {\mapsto} \mathsf{tt}], h''_2[l_2^x {\mapsto} (i+1)][l_2^b {\mapsto} \mathsf{tt}]) : W''_{\langle \mathsf{tt}, i+1\rangle}$. Moreover, and crucially, observe that from (9) we know $W''_{\langle \mathsf{tt}, i+1\rangle} \sqsupseteq^{\mathsf{pub}} W'$ because $\langle \mathsf{tt}, i+1\rangle$ is a public successor of $\langle \mathsf{tt}, i\rangle$. Finally, since $(W''_{\langle \mathsf{tt}, i+1\rangle}, \langle\rangle, \langle\rangle) \in \mathrm{V}[\![\mathsf{unit}]\!]$, the claim then follows from (6).

It remains to deal with related exceptions. We suppose (9)–(10) as above but then

11. $(W'', v_1', v_2') \in \mathrm{V}[\![\mathsf{exn}]\!]$

and have to show

$$(W'', K_1'[\mathsf{raise}\ v_1'; l_1^x := !l_1^x{+}1; l_1^b := \mathsf{tt}], K_2'[\mathsf{raise}\ v_2'; l_2^x := i{+}1; l_2^b := \mathsf{tt}]) \in \mathrm{O}.$$

By Lemma 4 it suffices to show $(W'', K_1'[\mathsf{raise}\ v_1'], K_2'[\mathsf{raise}\ v_2']) \in \mathrm{O}$, which follows from (6).

(B) We must show $(\widehat{W}, v_1^{\mathrm{poll}}, v_2^{\mathrm{poll}}) \in \mathrm{V}[\![\mathsf{unit} \to \mathsf{int}]\!]$. So we suppose

4. $W' \sqsupseteq \widehat{W}$
5. $(W', K_1', K_2') \in \mathrm{K}[\![\mathsf{int}]\!]$
6. $(h_1', h_2') : W'$
7. $\langle h_1'; K_1'[!l_1^x]\rangle \downarrow^{<W'.k}$

and need to show $\langle h_2'; K_2'[!l_2^x]\rangle \downarrow$. From (6) we know $h_1'(l_1^x) = h_2'(l_2^x) = i$ for some i. Consequently, we get $\langle h_1'; K_1'[i]\rangle \downarrow^{<W'.k}$ from (7). And similarly, if $\langle h_2'; K_2'[i]\rangle \downarrow$ then $\langle h_2'; K_2'[!l_2^x]\rangle \downarrow$. Since $(W', i, i) \in \mathrm{V}[\![\mathsf{int}]\!]$, the claim then follows from (5).

It turns out that the two programs are also equivalent in **FOSEC**. The proof is almost the same as the one for **HOSE**, except that a slightly different STS is used, and the part in (Ab) where we show

$$(W'_{\langle \mathsf{ff}, i\rangle}, K_1'[\bullet; l_1^x := !l_1^x + 1; l_1^b := \mathsf{tt}], K_2'[\bullet; l_2^x := i + 1; l_2^b := \mathsf{tt}]) \in \mathrm{K}[\![\mathsf{unit}]\!]$$

needs to be adapted.

The necessary change to the STS is, as shown in Figure 4, to remove the private transitions from states $\langle \mathsf{ff}, i\rangle$ to $\langle \mathsf{tt}, i+1\rangle$ (recall that the distinction between private and public transitions is not sound in the presence of call/cc). Assuming this is our island's STS, we now show

$$(W'_{\langle \mathsf{ff}, i\rangle}, K_1'[\bullet; l_1^x := !l_1^x + 1; l_1^b := \mathsf{tt}], K_2'[\bullet; l_2^x := i + 1; l_2^b := \mathsf{tt}]) \in \mathrm{K}[\![\mathsf{unit}]\!]$$

in the same environment as above. The part dealing with related exceptions stays unchanged.

In the other part, we only need to modify the argument establishing

$$(h_1''[l_1^x \mapsto (i+1)][l_1^b \mapsto \mathsf{tt}], h_2''[l_2^x \mapsto (i+1)][l_2^b \mapsto \mathsf{tt}]) : W''_{\langle \mathsf{tt}, i+1\rangle}.$$

Observe that this time $W''_{\langle \mathsf{tt}, i+1\rangle}$ is not a future world of W'' because our modified island has no transition from $\langle \mathsf{ff}, i\rangle$ to $\langle \mathsf{tt}, i+1\rangle$. However, *because heap relations in the **FOSEC** model are world-independent*, it is nevertheless easy to see from (10) that

$$(h_1''[l_1^x \mapsto (i+1)][l_1^b \mapsto \mathsf{tt}], h_2''[l_2^x \mapsto (i+1)][l_2^b \mapsto \mathsf{tt}]) : W''_{\langle \mathsf{tt}, i+1\rangle}$$

holds. The rest goes as before, in particular we still have $W''_{\langle \mathsf{tt}, i+1\rangle} \sqsupseteq^{\mathsf{pub}} W'$ because $\langle \mathsf{tt}, i+1\rangle$ is a successor of $\langle \mathsf{tt}, i\rangle$.

9.2 Deferred divergence (FOS version)

Recall the **FOS** version of the "deferred divergence" example from Section 6.2:

$$\begin{aligned}
\tau &= ((\mathsf{unit} \to \mathsf{unit}) \to \mathsf{unit}) \to \mathsf{unit}\\
e_1 &= \mathsf{let}\ y = \mathsf{ref\ ff\ in}\\
&\quad\ \lambda f.\, f\ (\lambda\_.\, y := \mathsf{tt});\\
&\qquad\ \mathsf{if}\ !y\ \mathsf{then}\ \bot\ \mathsf{else}\ \langle\rangle\\
e_2 &= \lambda f.\, f\ (\lambda\_.\, \bot)
\end{aligned}$$

We have already seen before that the two programs are neither equivalent in **FOSC** nor in **HOS**. They can also be distinguished in **FOSE**, for instance by the following context, which uses a callback that runs its argument and then raises an exception.

$$\bullet\ (\lambda g.\, (g\ \langle\rangle; \mathsf{raise}\ c))$$

As sketched in Section 6.2, e_1 and e_2 are equivalent in **FOS**, and, unsurprisingly, the proof combines all three of our model's special features (private transitions, inconsistent states, and backtracking). Here, we show the details of the difficult direction of approximation.

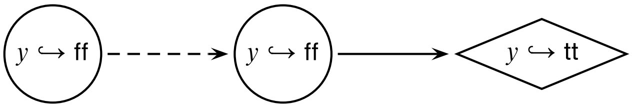
Formally, our goal is to prove $\cdot; \cdot; \cdot \vdash e_1 \precsim_{\log} e_2 : \tau$. Unfolding the definition, this reduces to showing $(W, e_1, e_2) \in \mathrm{E}[\![\tau]\!]$ for $W \in \mathrm{World}$. So we assume

1. $(W, K_1, K_2) \in \mathrm{K}[\![\tau]\!]$
2. $(h_1, h_2) : W$
3. $\langle h_1; K_1[e_1]\rangle \downarrow^{<W.k}$

and must show consistent(W) and $\langle h_2; K_2[e_2]\rangle \downarrow$. From (3) we get

$$\langle h_1 \uplus \{l_y \mapsto \mathsf{ff}\}; K_1[\widehat{e_1}[l_y/y]]\rangle \downarrow^{<W.k}$$

where $\widehat{e_1}$ is the body of the let-expression in e_1 and l_y some fresh location. We now extend the world with an island governing l_y and representing the STS from Section 6.2, with $s = 1$, 2, and 3 being the left, middle, and right states of the STS, respectively:


$y \hookrightarrow \mathsf{ff}$
$y \hookrightarrow \mathsf{ff}$
$y \hookrightarrow \mathsf{tt}$

$$\begin{aligned}
W_1 &= (W.k, (W.\Sigma_1, l_y{:}\mathsf{bool}), W.\Sigma_2, (W.\omega, \iota))\\
\iota &= (1, \delta, \varphi, \lightning, H)\\
\delta &= \{(1,2),(2,3)\}^*\\
\varphi &= \{(2,3)\}^*\\
\lightning &= \{3\}\\
H(1) &= \{(\widetilde{h_1}, \widetilde{h_2}) \mid \widetilde{h_1}(l_y) = \mathsf{ff}\}\\
H(2) &= \{(\widetilde{h_1}, \widetilde{h_2}) \mid \widetilde{h_1}(l_y) = \mathsf{ff}\}\\
H(3) &= \{(\widetilde{h_1}, \widetilde{h_2}) \mid \widetilde{h_1}(l_y) = \mathsf{tt}\}
\end{aligned}$$

Note that safe(ι) and therefore $W_1 \sqsupseteq^{\mathsf{pub}} W$. Using (2), it is also easy to see that $(h_1 \uplus \{l_y \mapsto \mathsf{ff}\}, h_2) : W_1$. Assuming we are able to show $(W_1, \widehat{e_1}[l_y/y], e_2) \in \mathrm{V}[\![\tau]\!]$, we can apply (1) to get consistent(W_1) and $\langle h_2; K_2[e_2]\rangle\downarrow$. The latter is one of the two things we needed to show. The other one is consistent(W). Since the only difference between W and W_1 is our island, this follows from consistent(W_1).

It remains to show $(W_1, \widehat{e_1}[l_y/y], e_2) \in \mathrm{V}[\![\tau]\!]$. So we suppose

4. $W' \sqsupseteq W_1$
5. $(W', f_1, f_2) \in \mathrm{V}[\![(\mathsf{unit} \to \mathsf{unit}) \to \mathsf{unit}]\!]$
6. $(W', K_1', K_2') \in \mathrm{K}[\![\mathsf{unit}]\!]$
7. $(h_1', h_2') : W'$
8. $e_1' = f_1\ (\lambda\_.\, l_y := \mathsf{tt}); \mathsf{if}\ !l_y\ \mathsf{then}\ \bot\ \mathsf{else}\ \langle\rangle$
9. $\langle h_1'; K_1'[e_1']\rangle \downarrow^{<W'.k}$

and need to show consistent(W') and $\langle h_2'; K_2'[f_2\ (\lambda\_.\,\bot)]\rangle\downarrow$. We now consider the case where our island's state in W' is 1—the other two are similar (and simpler). In that case, let $W_2' \sqsupseteq W'$ denote the world obtained from W' by transitioning our island's state from 1 to 2. Since the heap constraints of state 1 and 2 are the same, $(h_1', h_2') : W_2'$ follows immediately from (7). Now, we want to prove

$$(W_2', f_1\ (\lambda\_.\, l_y := \mathsf{tt}), f_2\ (\lambda\_.\,\bot)) \in \mathrm{E}[\![\mathsf{unit}]\!]$$

and

$$(W_2', K_1'[\bullet; \mathsf{if}\ !l_y\ \mathsf{then}\ \bot\ \mathsf{else}\ \langle\rangle], K_2') \in \mathrm{K}[\![\mathsf{unit}]\!],$$

as combining the two yields consistent(W_2') and $\langle h_2'; K_2'[f_2\ (\lambda\_.\,\bot)]\rangle\downarrow$. The latter is one of the two things we needed to show. The other one is consistent(W'), which obviously follows from consistent(W_2').

To show

$$(W_2', f_1\ (\lambda\_.\, l_y := \mathsf{tt}), f_2\ (\lambda\_.\,\bot)) \in \mathrm{E}[\![\mathsf{unit}]\!]$$

note that since $(W_2', f_1, f_2) \in \mathrm{V}[\![(\mathsf{unit} \to \mathsf{unit}) \to \mathsf{unit}]\!]$ by monotonicity, it suffices to show

$$(W_2', (\lambda\_.\, l_y := \mathsf{tt}), (\lambda\_.\,\bot)) \in \mathrm{V}[\![\mathsf{unit} \to \mathsf{unit}]\!].$$

Because the first function always terminates while the second never does, this is the part of the proof where inconsistency comes into play. We suppose

10. $W'' \sqsupseteq W_2'$
11. $(W'', K_1'', K_2'') \in \mathrm{K}[\![\mathsf{unit}]\!]$
12. $(h_1'', h_2'') : W''$
13. $\langle h_1''; K_1''[l_y := \mathsf{tt}]\rangle \downarrow^{<W''.k}$

and, with the help of the inconsistent state 3, will derive a contradiction. From (13) we get $\langle h_1''[l_y \mapsto \mathsf{tt}]; K_1''[\langle\rangle]\rangle \downarrow^{<W''.k}$. Let W_3'' denote the world obtained from W'' by transitioning our island's state to 3. Due to (10) we know that in W'' it is either 2 or 3, and thus we have $W_3'' \sqsupseteq^{\mathsf{pub}} W''$. Moreover, it is easy to see that $(h_1''[l_y \mapsto \mathsf{tt}], h_2'') : W_3''$. Since $(W_3'', \langle\rangle, \langle\rangle) \in \mathrm{V}[\![\mathsf{unit}]\!]$, instantiating (11) then yields consistent(W_3''), which is clearly in contradiction to 3 being an inconsistent state.

To show

$$(W_2', K_1'[\bullet; \mathsf{if}\ !l_y\ \mathsf{then}\ \bot\ \mathsf{else}\ \langle\rangle], K_2') \in \mathrm{K}[\![\mathsf{unit}]\!],$$

we suppose

10. $W'' \sqsupseteq^{\mathsf{pub}} W_2'$
11. $(h_1'', h_2'') : W''$
12. $\langle h_1''; K_1'[\mathsf{if}\ !l_y\ \mathsf{then}\ \bot\ \mathsf{else}\ \langle\rangle]\rangle \downarrow^{<W''.k}$

and must show consistent(W'') and $\langle h_2''; K_2'[\langle\rangle]\rangle \downarrow$. From (12) it is clear that $h_1''(l_y)$ must be ff. This implies that $\langle h_1''; K_1'[\langle\rangle]\rangle \downarrow^{W''.k}$ and that our island's state in W'' must be 2. We now want to instantiate (6), but W'' does not publicly extend W' because there is no public transition from state 1 to state 2 (recall that we are considering the case where our island in W' is in state 1).

However, we can now *backtrack* to state 1: let W_1'' denote the world obtained from W'' by setting our island's state from 2 to 1. Because both states express the same heap constraint and because heap relations for **FOS** are world-independent, $(h_1'', h_2'') : W_1''$ follows from just (11). Note that $W'' \sqsupseteq^{\mathsf{pub}} W_2'$ implies $W_1'' \sqsupseteq^{\mathsf{pub}} W'$. Finally, we can use (6) with $(W_1'', \langle\rangle, \langle\rangle) \in \mathrm{V}[\![\mathsf{unit}]\!]$, and thus obtain consistent(W_1'') and $\langle h_2''; K_2'[\langle\rangle]\rangle \downarrow$. Since state 2 is consistent, the former implies consistent(W_2'') and we are done.

9.3 One-shot continuations

This example, due to Friedman & Haynes (1985), demonstrates how call/cc can be encoded in **HOSC** using local state and one-shot continuations, where the latter are themselves encoded using call/cc and a local ref cell.

Let $\tau_\alpha = \mathsf{cont}\,\alpha \to \alpha$. We start by turning call/cc into a value of type $\forall\alpha.\tau_\alpha \to \alpha$:

$$\mathsf{callcc} \overset{\text{def}}{=} \Lambda\alpha.\lambda f{:}\tau_\alpha.\,\mathsf{call/cc}_\alpha(x.\ f\ x)$$

Let us now define callcc1, its one-shot version:

$$\begin{aligned}\mathsf{callcc1} \overset{\text{def}}{=} \Lambda\alpha.\lambda f{:}\tau_\alpha.\,&\mathsf{let}\ b = \mathsf{ref\ ff\ in}\\ &\mathsf{call/cc}_\alpha(x.\\ &f\ (\mathsf{cont}_\alpha\ (\mathsf{let}\ y = \bullet\ \mathsf{in\ if}\ !b\ \mathsf{then}\ \bot_{\mathsf{unit}}\\ &\qquad\qquad\qquad \mathsf{else}\ (b := \mathsf{tt}; \mathsf{throw}_{\mathsf{unit}}\ y\ \mathsf{to}\ x)))))\end{aligned}$$

Compare this to callcc. callcc1 first creates a reference holding a boolean flag that expresses whether the continuation has been used (the shot has been fired) already. Correspondingly, the continuation it passes to f is not just x but rather a wrapper around x that diverges if it has been used before (otherwise it behaves like x but flips the flag).

The encoding of call/cc using one-shot continuations, callcc′, is as follows:

$$\mathsf{callcc}' \overset{\text{def}}{=} \Lambda\alpha.\lambda f{:}\tau_\alpha.\,\mathsf{let}\ r = \mathsf{ref}\ (\mathsf{cont}_\alpha\ \bullet)\ \mathsf{in}\ G_r^\alpha\ f$$

which makes use of the following abbreviation[7]:

$$\begin{aligned}G_r^\alpha \overset{\text{def}}{=} \mathsf{fix}\ g(f).\ &\mathsf{let}\ x = \mathsf{callcc1}\ \alpha\ (\lambda z{:}\mathsf{cont}\,\alpha.\,(r := z; f\ (\mathsf{cont}_\alpha\ (\mathsf{throw}_{\mathsf{unit}}\ \bullet\ \mathsf{to}\ !r))))\ \mathsf{in}\\ &\mathsf{callcc1}\ \alpha\ (\lambda z{:}\mathsf{cont}\,\alpha.\,g\ (\lambda\_{:}\mathsf{cont}\,\alpha.\,\mathsf{throw}_{\mathsf{unit}}\ x\ \mathsf{to}\ z))\end{aligned}$$

[7] Here, fix $f(x).e$ encodes a recursive function in the usual way using recursive types [details can be found in the technical appendix (Dreyer *et al.*, 2012)].

The code is quite tricky and hard to explain. The correctness proof, however, is conceptually simple as it requires only a single (permanent) invariant. In fact, we will show that callcc is equivalent to callcc' in **HOSEC**, i.e., even in the presence of exceptions.

We show $\cdot;\cdot;\cdot \vdash \mathsf{callcc} \precsim_{\log} \mathsf{callcc}' : \forall\alpha.\, \tau_\alpha \to \alpha$ (the other direction is similar), which immediately reduces to showing

$$(W_0, \mathsf{callcc}, \mathsf{callcc}') \in \mathrm{V}[\![\forall\alpha.\, \tau_\alpha \to \alpha]\!]$$

for $W_0 \in \mathrm{World}$. So suppose $W_1 \sqsupseteq W_0$ and $(\tau_1, \tau_2, r) \in \mathrm{SomeValRel}$. We must show that

$$(W_1, \lambda f.\, \mathsf{call/cc}\,(x.\ f\ x), \lambda f.\, \mathsf{let}\ r = \mathsf{ref}\ (\mathsf{cont}\ \bullet)\ \mathsf{in}\ G_r^{\tau_2}\ f) \in \mathrm{V}[\![\tau_\alpha \to \alpha]\!]\rho$$

where $\rho = \alpha \mapsto (\tau_1, \tau_2, r)$.

To do so, we suppose

1. $W \sqsupseteq W_1$
2. $(W, v_1, v_2) \in \mathrm{V}[\![\tau_\alpha]\!]\rho$
3. $(W, K_1, K_2) \in \mathrm{K}[\![\alpha]\!]\rho$
4. $(h_1, h_2) : W$
5. $\langle h_1 ; K_1[\mathsf{call/cc}\,(x.\ v_1\ x)]\rangle \downarrow^{<W.k}$

and have to show $\langle h_2 ; K_2[\mathsf{let}\ r = \mathsf{ref}\ (\mathsf{cont}\ \bullet)\ \mathsf{in}\ G_r^{\tau_2}\ v_2]\rangle \downarrow$. From (5) we know that

$$\langle h_1 ; K_1[v_1\ (\mathsf{cont}\ K_1)]\rangle \downarrow^{<W.k}.$$

Similarly, unfolding $G_r^{\tau_2}$ makes it easy to verify that $\langle h_2 ; K_2[\mathsf{let}\ r = \mathsf{ref}\ (\mathsf{cont}\ \bullet)\ \mathsf{in}\ G_r^{\tau_2}\ v_2]\rangle \downarrow$ if $\langle \widehat{h_2} ; K_2[v_2\ (\mathsf{cont}\ (\mathsf{throw}\ \bullet\ \mathsf{to}\ !l_r))]\rangle \downarrow$, where

- $\widehat{h_2} = h_2 \uplus \{l_r \mapsto e_{l_b}\} \uplus \{l_b \mapsto \mathsf{ff}\}$ (for distinct and fresh l_r, l_b)
- $e_l = \mathsf{cont}\ (\mathsf{let}\ y = \bullet\ \mathsf{in}\ \mathsf{if}\ !l\ \mathsf{then}\ \bot\ \mathsf{else}\ (l := \mathsf{tt}; \mathsf{throw}\ y\ \mathsf{to}\ \mathsf{cont}\ K))$
- $K = K_2[\mathsf{let}\ x = \bullet\ \mathsf{in}\ \mathsf{callcc1}\ \tau_2\ (\lambda z.\, G_{l_r}^{\tau_2}\ (\lambda\_.\, \mathsf{throw}\ x\ \mathsf{to}\ z))]$

We now extend the world with an island concerning l_r and l_b. Let

- $\widehat{W} = (W.k, W.\Sigma_1, (W.\Sigma_2, l_r{:}\mathsf{cont}\,\tau_2, l_b{:}\mathsf{bool}), (W.\omega, \iota))$
- $\iota = (\langle\rangle, \emptyset^*, \emptyset^*, \emptyset, H)$
- $H(\langle\rangle) = \{(\widetilde{W}, \widetilde{h_1}, \widetilde{h_2}) \in \mathrm{HeapAtom} \mid \exists l \in \mathrm{dom}(h_2).\ \widetilde{h_2}(l_r) = e_l \wedge \widetilde{h_2}(l) = \mathsf{ff}\}$

It is easy to see that $\widehat{W} \sqsupseteq W$ and then that $(h_1, \widehat{h_2}) : \widehat{W}$. By (3) and monotonicity it thus suffices to show

$$(\widehat{W}, v_1\ (\mathsf{cont}\ K_1), v_2\ (\mathsf{cont}\ (\mathsf{throw}\ \bullet\ \mathsf{to}\ !l_r))) \in \mathrm{E}[\![\alpha]\!]\rho.$$

For this, it in turn suffices, due to (2) and Lemma 4, to show

$$(\widehat{W}, \mathsf{cont}\ K_1, \mathsf{cont}\ (\mathsf{throw}\ \bullet\ \mathsf{to}\ !l_r)) \in \mathrm{V}[\![\mathsf{cont}\,\alpha]\!]\rho.$$

To this end, we suppose

6. $W' \sqsupseteq \widehat{W}$
7. $(W', v_1', v_2') \in \mathrm{V}[\![\alpha]\!]\rho$
8. $(h_1', h_2') : W'$

9. $\langle h_1' ; K_1[v_1'] \rangle \downarrow^{<W'.k}$

and need to show $\langle h_2' ; \mathsf{throw}\ v_2'\ \mathsf{to}\ !l_r \rangle \downarrow$. From $(h_1', h_2') : W'$ we know that $h_2'(l_r) = e_l$ and $h_2'(l) = \mathsf{tt}$, for some l. Hence

$$\begin{array}{l} \langle h_2' ; \mathsf{throw}\ v_2'\ \mathsf{to}\ !l_r \rangle \downarrow \ \text{ if} \\ \langle h_2'[l \mapsto \mathsf{ff}] ; K_2[\mathsf{callcc1}\ \tau_2\ (\lambda z. G_{l_r}^{\tau_2}\ (\lambda\_. \mathsf{throw}\ v_2'\ \mathsf{to}\ z))] \rangle \downarrow \ \text{ if} \\ \langle h_2'[l \mapsto \mathsf{ff}][l_r \mapsto e_{l'}] \uplus \{l' \mapsto \mathsf{tt}\} ; K_2[v_2'] \rangle \downarrow \end{array}$$

where l' is fresh. Notice that l' now has taken over the role of l. So let

$$W'' = (W'.k, W'.\Sigma_1, (W'.\Sigma_2, l' {:} \mathsf{bool}), W'.\omega).$$

Since $W'' \sqsupseteq W'$ we get, using monotonicity, that

$$(h_1', h_2'[l \mapsto \mathsf{ff}][l_r \mapsto e_{l'}] \uplus \{l' \mapsto \mathsf{tt}\}) : W''.$$

Finally, we use (3) and (6) to instantiate (7), which yields the claim.

9.4 Well-bracketed state change

Recall the motivating example from Section 5.

$$\begin{array}{ll} \tau & = (\mathsf{unit} \to \mathsf{unit}) \to \mathsf{int} \\ e_1 & = \mathsf{let}\ x = \mathsf{ref}\ 0\ \mathsf{in} \\ & \qquad \lambda f. (x := 0; f\ \langle\rangle; x := 1; f\ \langle\rangle; !x) \\ e_2 & = \lambda f. (f\ \langle\rangle; f\ \langle\rangle; 1) \end{array}$$

We have already shown in Section 5 that e_1 and e_2 are not equivalent in **HOSC** (and **FOSC**). It is also easy to see that they are not equivalent in **HOSE** (or even **FOSE**). The following is a distinguishing context C:

$$\begin{array}{l} \mathsf{let}\ g = \bullet\ \mathsf{in} \\ \mathsf{let}\ f_2 = (\lambda\_. \mathsf{raise}\ c)\ \mathsf{in} \\ \mathsf{let}\ x = \mathsf{ref}\ \mathsf{tt}\ \mathsf{in} \\ \mathsf{let}\ f_1 = (\lambda\_. \mathsf{if}\ !x\ \mathsf{then}\ x := \mathsf{ff}\ \mathsf{else}\ \mathsf{try}\ g\ f_2\ \mathsf{with}\ \_.\langle\rangle)\ \mathsf{in} \\ g\ f \end{array}$$

Again, it is easy to verify that $C[e_1]$ yields 0, while $C[e_2]$ yields 1.

The programs are, however, equivalent in **HOS**, which we can prove by showing that each logically approximates the other. Here we only present one direction, the other is similar.

We show $\cdot;\cdot;\cdot \vdash e_1 \lesssim_{\log} e_2 : (\mathsf{unit} \to \mathsf{unit}) \to \mathsf{int}$, which immediately reduces to showing $(W, e_1, e_2) \in \mathrm{E}[\![(\mathsf{unit} \to \mathsf{unit}) \to \mathsf{int}]\!]$ for $W \in \mathrm{World}$. To do so, we suppose

1. $(W, K_1, K_2) \in \mathrm{K}[\![(\mathsf{unit} \to \mathsf{unit}) \to \mathsf{int}]\!]$
2. $(h_1, h_2) : W$
3. $\langle h_1 ; K_1[e_1] \rangle \downarrow^{<W.k}$

and must now show $\langle h_2 ; K_2[e_2] \rangle \downarrow$. From (3) we know that

$$\langle h_1 \uplus \{l_x \mapsto 0\} ; K_1[\lambda f. (l_x := 0; f\ \langle\rangle; l_x := 1; f\ \langle\rangle; !l_x)] \rangle \downarrow^{<W.k}$$

for some fresh l_x. We will now extend the world with an island for l_x and then show that the above function value is related to e_2 in that world. Let

- $W_0 = (W.k, (W.\Sigma_1, l_x{:}\mathsf{int}), W.\Sigma_2, (W.\omega, \iota))$
- $\iota = (0, \delta, \varphi, \emptyset, H)$
- $\delta = (\varphi \uplus \{(1,0)\})^*$
- $\varphi = \{(0,1)\}^*$
- $H(i) = \{(\widetilde{W}, \widetilde{h}_1, \widetilde{h}_2) \in \mathrm{HeapAtom} \mid \widetilde{h}_1(l_x) = i\}$

W_0 publicly extends W by the following STS (with its left state being the current one):

$$\begin{tikzcd} x \hookrightarrow 0 \arrow[r, bend left] & x \hookrightarrow 1 \arrow[l, bend left, dashed] \end{tikzcd}$$

Using monotonicity it is then easy to see that $(h_1 \uplus \{l_x \mapsto 0\}, h_2) : W_0$. If we can show

$$(W_0, \lambda f.\,(l_x := 0; f\ \langle\rangle; l_x := 1; f\ \langle\rangle; !l_x), e_2) \in \mathrm{V}[\![(\mathsf{unit} \to \mathsf{unit}) \to \mathsf{int}]\!]$$

then instantiating (1) yields the claim.

So we suppose

4. $W' \sqsupseteq W_0$
5. $(W', \lambda z.\,e_1', \lambda z.\,e_2') \in \mathrm{V}[\![\mathsf{unit} \to \mathsf{unit}]\!]$
6. $(W', K_1', K_2') \in \mathrm{K}[\![\mathsf{int}]\!]$
7. $(h_1', h_2') : W'$
8. $\langle h_1'; K_1'[(l_x := 0; (\lambda z.\,e_1')\ \langle\rangle; l_x := 1; (\lambda z.\,e_1')\ \langle\rangle; !l_x)]\rangle \downarrow^{<W'.k}$

and have to show $\langle h_2'; K_2'[((\lambda z.\,e_2')\ \langle\rangle; (\lambda z.\,e_2')\ \langle\rangle; 1)]\rangle \downarrow$. From (8) we know

$$\langle h_1'[l_x \mapsto 0]; K_1'[(e_1'[\langle\rangle/z]; l_x := 1; (\lambda z.\,e_1')\ \langle\rangle; !l_x)]\rangle \downarrow^{<W'.k}.$$

Let W_0' denote the world obtained from W' by setting our island's current state (which may be either 0 or 1) to 0. It is easy to see that $W_0' \sqsupseteq W'$ and thus $(h_1'[l_x \mapsto 0], h_2') : W_0'$. Since $(W_0', \langle\rangle, \langle\rangle) \in \mathrm{V}[\![\mathsf{unit}]\!]$, instantiating (5) yields

$$(W_0', e_1'[\langle\rangle/z], e_2'[\langle\rangle/z]) \in \mathrm{E}[\![\mathsf{unit}]\!]\,.$$

Hence, if we can now show

$$(W_0', K_1'[(\bullet; l_x := 1; (\lambda z.\,e_1')\ \langle\rangle; !l_x)], K_2'[(\bullet; (\lambda z.\,e_2')\ \langle\rangle; 1)]) \in \mathrm{K}[\![\mathsf{unit}]\!]$$

then we get

$$\langle h_2'; K_2'[(e_2'[\langle\rangle/z]; (\lambda z.\,e_2')\ \langle\rangle; 1)]\rangle \downarrow,$$

which implies $\langle h_2'; K_2'[((\lambda z.\,e_2')\ \langle\rangle; (\lambda z.\,e_2')\ \langle\rangle; 1)]\rangle \downarrow$

To this end, we suppose

9. $W'' \sqsupseteq^{\mathsf{pub}} W_0'$
10. $(h_1'', h_2'') : W''$
11. $\langle h_1''; K_1'[(\langle\rangle; l_x := 1; (\lambda z.\,e_1')\ \langle\rangle; !l_x)]\rangle \downarrow^{<W''.k}$

and have to show $\langle h_2''; K_2'[(\langle\rangle; (\lambda z. e_2')\ \langle\rangle; 1)]\rangle\downarrow$. From (11) we know

$$\langle h_1''[l_x \mapsto 1]; K_1'[(e_1'[\langle\rangle/z]; !l_x)]\rangle \downarrow^{<W''.k}.$$

Let W_1'' denote the world obtained from W'' by setting our island's current state (which may again be either 0 or 1) to 1. It is easy to see that W_1'' *publicly* extends W'' and thus $(h_1''[l_x \mapsto 1], h_2'') : W_1''$. Since $(W_1'', \langle\rangle, \langle\rangle) \in \mathrm{V}[\![\mathsf{unit}]\!]$, instantiating (5) yields

$$(W_1'', e_1'[\langle\rangle/z], e_2'[\langle\rangle/z]) \in \mathrm{E}[\![\mathsf{unit}]\!].$$

Hence, if we can now show

$$(W_1'', K_1'[(\bullet; !l_x)], K_2'[(\bullet; 1)]) \in \mathrm{K}[\![\mathsf{unit}]\!]$$

then we get $\langle h_2'; K_2'[(e_2'[\langle\rangle/z]; 1)]\rangle\downarrow$, which implies $\langle h_2'; K_2'[((\lambda z. e_2')\ \langle\rangle; 1)]\rangle\downarrow$

To this end, we suppose

12. $W''' \sqsupseteq^{\mathsf{pub}} W_1''$
13. $(h_1''', h_2''') : W'''$
14. $\langle h_1'''; K_1'[(\langle\rangle; !l_x)]\rangle \downarrow^{<W'''.k}$

and must show $\langle h_2'''; K_2'[(\langle\rangle; 1)]\rangle\downarrow$. Observe that (12) and (13) imply $h_1'''(l_x) = 1$ (there is no public transition leading out of the $x \hookrightarrow 1$ state). Therefore, we know from (14) that $\langle h_1'''; K_1'[1]\rangle \downarrow^{<W'''.k}$. Independently, it is not hard to see that (9) and (12) imply $W''' \sqsupseteq^{\mathsf{pub}} W'$ (the $x \hookrightarrow 1$ state is publicly reachable from whatever state our island was in in W'). As $(W''', 1, 1) \in \mathrm{V}[\![\mathsf{int}]\!]$, instantiating (6) yields $\langle h_2'''; K_2'[1]\rangle\downarrow$, which in turn implies the claim.

9.5 Single return

This example is due to Thielecke (2000) (see Section 4 of his paper). His proof method is relatively brute-force, however, and we can easily prove his example using an STS with a single private transition.

$$\begin{aligned}
\tau\ &= (\mathsf{unit} \to \mathsf{unit}) \to \mathsf{int}\\
e_1 &= \lambda f.\, \mathsf{let}\ x = \mathsf{ref}\ 0\ \mathsf{in}\ \mathsf{let}\ y = \mathsf{ref}\ 0\ \mathsf{in}\\
&\qquad f\ \langle\rangle; x := !y; y := 1; !x\\
e_2 &= \lambda f.\, \mathsf{let}\ x = \mathsf{ref}\ 0\ \mathsf{in}\ \mathsf{let}\ y = \mathsf{ref}\ 0\ \mathsf{in}\\
&\qquad f\ \langle\rangle; x := !y; y := 2; !x
\end{aligned}$$

The programs are not equivalent in **HOSC** as the following context C demonstrates:

$$\begin{aligned}
&\mathsf{let}\ g = \bullet\ \mathsf{in}\\
&\mathsf{call/cc}\,(k_0.\ \mathsf{let}\ r = \mathsf{ref}\ k_0\ \mathsf{in}\\
&\qquad\qquad \mathsf{let}\ b = \mathsf{ref}\ \mathsf{tt}\ \mathsf{in}\\
&\qquad\qquad \mathsf{let}\ f = (\lambda\_.\, \mathsf{call/cc}\,(k.\ r := k))\ \mathsf{in}\\
&\qquad\qquad \mathsf{let}\ x = g\ f\ \mathsf{in}\\
&\qquad\qquad \mathsf{if}\ !b\ \mathsf{then}\ b := \mathsf{ff}; \mathsf{throw}\ \langle\rangle\ \mathsf{to}\ !r\ \mathsf{else}\ x)
\end{aligned}$$

It is easy to verify that $C[e_1]$ yields 1, while $C[e_2]$ yields 2.

They are, however, equivalent in **HOSE**, for which we now sketch the proof. Assuming e_1 and e_2 are applied in some world W, we symbolically execute the allocations and add the following STS to W, making the left state its current state:

$$\begin{tikzcd} \boxed{y \hookrightarrow 0} \arrow[r, dashed] & \boxed{x \hookrightarrow 0} \end{tikzcd}$$

When the two programs now invoke their callback argument, we know that, assuming they return, they do so still in the left state, as there is no public transition leading out of it. Hence x will be set to 0 in both programs, after which we transition to the right state and observe that both return 0. Note that despite this private transition, the resulting world *publicly* extends the initial world W—because W did not contain the island in question at all, and any extension of W with a new island is considered a public extension—so we are done.

9.6 Higher-order profiling

The following example is due to Pitts & Stark (1998). Although it seems quite different in nature from the previous one, it turns out that it can be proved using an STS structurally equivalent to the one above.

$$\begin{aligned}
\tau &= \tau_f \to \tau_g \to \tau_f \to \mathsf{int} \\
e_1 &= \lambda f.\, \lambda g.\, \lambda f'.\, \mathsf{let}\ \langle g', g'' \rangle = p\ g\ \mathsf{in}\ (f\ g'; g'\ \langle\rangle; f'\ g'; g''\ \langle\rangle) \\
e_2 &= \lambda f.\, \lambda g.\, \lambda f'.\, \mathsf{let}\ \langle g', g'' \rangle = p\ g\ \mathsf{in}\ (f\ g'; g\ \langle\rangle; f'\ g'; g''\ \langle\rangle + 1)
\end{aligned}$$

where

$$\begin{aligned}
\tau_f &= \tau_g \to \mathsf{unit} \\
\tau_g &= \mathsf{unit} \to \mathsf{unit} \\
p &= \lambda g.\, \mathsf{let}\ c = \mathsf{ref}\ 0\ \mathsf{in}\ \langle \lambda x.\, (c := {!c} + 1; g\ x), \lambda\_.\, {!c} \rangle
\end{aligned}$$

The higher-order function p takes a function g and returns a profiling version of it, i.e., a function that behaves like g except that each time it is called it also increments a local counter by 1. Additionally, p returns a function for reading the current value of that counter. We sketch the proof that e_1 and e_2 are equivalent, which can be seen as a partial correctness proof of the profiling operation.

We assume the two functions are called in an initial world W. When the profiling operation p is applied, we add the following STS to W:

$$\begin{tikzcd} \boxed{\frac{c \hookrightarrow n}{c \hookrightarrow n}} \arrow[r, dashed] & \boxed{\frac{c \hookrightarrow n+1}{c \hookrightarrow n}} \end{tikzcd}$$

The left state, which we make the current state, asserts that the counter c has the same value in both programs (this is fine because at that time it points to 0 in both). The right one asserts that the value stored at c in the first program is 1 greater than in the second program.

Consequently, if and when f returns, we know that we are still in the left state as there is no public transition leading anywhere else. (Of course c might have increased by now, but if so, then by the same amount on both sides: the only way f can touch the counter is by running its argument.) Now we call the tracked version of g in the first program and the original g in the second, thus incrementing the counter only in the first. Accordingly, we move along the private transition to the right state. Then f' is called and, since there is no other state reachable from here, c in the first program will still point to a number 1 greater than in the second program if and when f' returns. Consequently, $g'' \langle\rangle$ in the first program will yield the same value as $g'' \langle\rangle + 1$ in the second. Moreover, the world at that point is a public future world of W.

10 Related and future work

Many techniques have been proposed for reasoning about contextual equivalence of stateful programs. Using a variety of these techniques, most of the examples we discuss in this paper *have* been proved already (with minor variations) in different language settings, but there has not heretofore been any clear account of how they all fit together. Indeed, our main contribution lies in our unifying framework of STSs, along with the realization that the absence of call/cc and/or higher-order state enables the extension of our STS model in orthogonal ways. That said, some of our examples are also new, such as "callback with lock" in **FOSEC**, and the other ADR examples in **HOSEC**.

Game semantics. As explained in the Introduction, game semantics has served as an inspiration to us, especially Abramsky's idea of the "semantic cube." There are many papers on this topic; perhaps the two most relevant to our present work are Laird's model of call-by-name PCF extended with a control operator (Laird, 1997) and Abramsky, Honda, and McCusker's model of call-by-value PCF extended with general references (Abramsky *et al.*, 1998). In the latter, references are modeled essentially as arbitrary pairs of "reading" and "writing" functions. Unfortunately, this means that there are bogus references ("bad variables"), which do not behave like regular references and thus break some basic equivalences. It also means that this model—unlike ours—cannot support pointer equality.[8] In very recent follow-up work, Murawski & Tzevelekos (2011) managed to overcome these issues.

The primary focus of the research on games models has been full abstraction. One of the key motivations for having a fully abstract model is, of course, that it allows one to prove two programs observationally equivalent by proving that their denotations (in games models, "strategies") are the same. However, the games models do not in general directly facilitate such proofs since the strategies are non-trivial to analyze for equality (and since game categories also involve a non-trivial quotienting). Hence, proof methods for proving actual program equivalences

[8] We have not emphasized the fact that we model pointer equality in this paper, but some of ADR's examples do make use of it, and it is a feature one generally expects to find in real ML-like languages.

based on specific games models have primarily been developed only for simple languages with state, namely call-by-name Idealized Algol. For a finitary version of that language (i.e., a version with only finite ground types and no recursion) there is a full classification of when contextual equivalence is decidable (e.g., see Ghica & McCusker, 2000; Murawski & Walukiewicz, 2008). A finitary version of a call-by-value variant has also been studied by Murawski (2005), and with that model he could show some finitary versions of the examples of Pitts and Stark, e.g., the profiling example (see p. 29 of Murawski, 2005).

Another focus of game semantics is on understanding how the presence of different features in a language affects the kinds of interactions a program can have with its context. Laird (1997) models the presence of control operators by relaxing the "well-bracketing" restriction on strategies. Abramsky *et al.* (1998) model the presence of higher-order state by relaxing the "visibility" restriction. There seems intuitively to be some correspondence between the former and our private transitions, and between the latter and our backtracking, but determining the precise nature of this correspondence is left to future work.

Operational game semantics. Another line of related work concerns what some have called "operational game semantics." This work considers labeled transition systems, and either traces or bisimulation relations over those, directly inspired by games models. Such so-called "normal form bisimulation" relations have been developed for an untyped language with state and control (Støvring & Lassen, 2007), for a typed language with recursive types (but no state) (Lassen & Levy, 2007), and for a language with impredicative polymorphism (but no state) (Lassen & Levy, 2008). Laird (2007) gave a fully abstract trace semantics for the language of Abramsky *et al.* (1998) extended with pointer equality. His trace-sets may be viewed as deterministic strategies in the sense of game semantics. Normal form bisimulations have been used to prove contextual equivalence of actual examples, e.g., Støvring and Lassen's proof of correctness (Støvring & Lassen, 2007) for the encoding of call/cc via one-shot continuations that we described at the end of Section 4. Koutavas and Lassen have shown, in unpublished work (Koutavas & Lassen, 2008), how Laird's trace semantics can be used to prove the **HOS** version of the deferred divergence example (Section 5.2), by showing that the two programs have the same set of traces.

It is difficult to directly compare the proofs of these examples using operational game semantics methods versus the proofs using our present model; although the essential proof ideas are the similar, their formalizations are very different, and neither approach is clearly superior. However, to the best of our knowledge, no fully abstract games model (either operational or denotational) has yet been given for the rich language that we consider in this paper (call-by-value, impredicative polymorphism, general references with pointer equality, call/cc, and recursive types).

Logical relations. Our work is heavily indebted to the pioneering work of Pitts & Stark (1998), who gave a fully abstract logical relation for a simply-typed functional language with recursion and first-order state. In particular, we rely on the basic setup of their biorthogonal Kripke model, although (like ADR's) ours is also step-indexed.

In the absence of step indices, biorthogonality renders the logical relation *admissible* (an important property when modeling recursion). In the presence of step indices, admissibility is not as important, since the model essentially only consists of finite approximations, and there is no need to ever talk about their limit. Nevertheless, as we have seen, biorthogonality plays a crucial role in modeling control and ensuring full abstraction.

With respect to the latter, it is not clear how useful the full abstraction property is for us *per se*, since it is achieved in a largely "feature-independent" manner. That is, the proof that biorthogonality makes the logical relation complete is essentially the same for each of the four languages we consider, so full abstraction here is perhaps not the most informative criterion. One could for instance take Pitts and Stark's original model, add step-indexing to it, and get out a different fully abstract model for **HOSC**. Clearly, that model would not be as practically powerful as our STS-based model, but it would nevertheless be fully abstract.

Aside from ADR, the closest logical relations to ours are the ones developed by Bohr in her thesis (Bohr, 2007). Hers also employ biorthogonality, albeit in a denotational setting. Her possible worlds bear some similarity to ADR's in that they, too, allow one to model heap properties that evolve over time. In addition, they allow one to impose constraints on continuations. Like us, she is also able to handle the **HOS** version of the deferred divergence example, but the language she considers is not as rich as ours (it does not support full polymorphism), and she does not consider handling call/cc or the restriction to first-order state. We can prove all of the examples from her thesis, and we believe that our proofs are significantly simpler to understand.

Regarding the deferred divergence example: it is originally due to O'Hearn, who formulated it in the context of Idealized Algol (O'Hearn & Reddy, 1995). Pitts showed how to prove this example using operational Kripke logical relations, by allowing the parameters of the logical relation to relate proper states to undefined states (i.e., by phrasing heap relations over "lifted" heaps) (Pitts, 1996). It is not clear whether this technique generalizes to higher-order state, however.

More recently, Johann *et al.* (2010) have proposed a generic framework for operational reasoning about algebraic effects. Their work is complementary to ours: they develop effect-independent proof principles, whereas we develop effect-specific proof principles. They do not consider local state, higher-order state, or control.

Our decision to employ both step-indexing and biorthogonality was influenced directly by the work of Benton, together with Tabareau (Benton & Tabareau, 2009) and Hur (Benton & Hur, 2009), on compiler correctness. They argue persuasively for the benefits of combining the two techniques.

Birkedal *et al.* (2011) recently proposed a new type theory and logic for guarded recursion, which allows for the formation of both guarded recursive types and guarded recursive predicates. The former means that the type theory is sufficiently expressive to allow for the construction of recursive worlds used in step-indexed models. Guarded recursive predicates are used to define the operational semantics of the programming language. Thence it is possible to give a more abstract "synthetic" logical relations model where the step-indexing is hidden and replaced by a few uses of guarded-recursion operators. In Birkedal *et al.* (2011) this approach was demonstrated for a simple unary model of HOS; we believe the approach scales well and that it should be possible to apply it to construct the models considered in the present paper.

Lastly, since the original conference version of this paper was published, Reddy & Dunphy (2011) have proposed a relational model that accounts for a number of our examples in a denotational setting (albeit focused on the setting of Idealized Algol). Their approach to local reasoning is essentially to allow an object to impose restrictions on (1) the set of possible states its private fields can be in, and (2) the set of possible transformations between those states. At first glance, this seems roughly similar to our method of state transition systems, but limited to a single notion of transition. However, by exploiting relational parametricity in their denotational model, Reddy and Dunphy are also able to account for examples—such as our well-bracketed state change example—that in our model demand a combination of public and private transitions. More concrete examples remain to be worked out in order to gain a clearer understanding of the practical expressive power of their model as well as its relationship to ours.

Environmental bisimulations and relation transition systems. For reasoning about contextual equivalences (involving either type abstraction or local state), one of the most successful alternatives to logical relations is the coinductive technique of *environmental (aka "relation-sets") bisimulations*. The current state of the art is Sumii's work on type abstraction and general references (Sumii, 2009), which builds on work by Sumii & Pierce (2007), Koutavas & Wand (2006), and Sangiorgi *et al.* (2011). Sumii is able to handle all the examples we have presented here in the setting of **HOS**; he does not consider call/cc or first-order state (but does, in Sangiorgi *et al.*, 2011, consider concurrency). Being coinductive, his proofs avoid the tedium of reasoning about step-indexing, but in some cases (e.g., for the well-bracketed version of the "awkward" example—see Section 5.1), they are somewhat "brute-force" in the sense that they require explicit reasoning about the intensional structure of program contexts.

Essentially, it seems that there is a close correspondence between environmental bisimulation proofs and proofs using our method that only rely on the use of *public* transitions. Specifically, environmental bisimulations are defined as *sets* X of relations, and one can roughly think of them as baking in a fixed state transition system, in which the states are the relations R in X, and a state (i.e., relation) R' is (both publicly and privately) accessible from R iff $R' \supseteq R$. Environmental bisimulations' apparent conflation of public and private transitions would explain why they do not offer a clean method for proving the well-bracketed "awkward" example, which requires a distinction between the two forms of transition. In contrast, our state transition systems capture the intuitions about well-bracketing more directly. That said, it seems quite plausible that the environmental bisimulation method could be generalized to support a richer private/public distinction along the lines of our present model.

Indeed, Hur *et al.* (2012) have recently presented a new type of model—dubbed *relation transition systems (RTSs)*—which achieves something in this general direction. RTSs marry together the coinductive style of bisimulations with a treatment of local, higher-order state (via public and private transitions) closely based on the framework we have presented in this paper. Proofs using RTSs are very similar in their core content to proofs using our method; the main differences between the methods are structural. In particular, RTS proofs have a very rigid structure—inspired in part by that of normal form bisimulation proofs (discussed above)—which makes them easier to compose transitively than logical-relations proofs, but at the expense of a more complicated definition of the model itself. Thus far, RTSs have only been used to model the **HOS** language, and (unlike logical-relations methods) they fail to validate η-equivalence for function values.

Anti-frame rule. Pottier (2008) has proposed an alternative way of reasoning about local state using a rich type system with capabilities, regions, and linearity. His *anti-frame rule* allows one to establish a hidden property about a piece of local state, much in the same way that our islands do. In its original form, however, the anti-frame rule was restricted to reasoning about *invariants*, which we argued in Section 3 are insufficient for many examples.

To address this limitation, Pottier has suggested two extensions of his framework. First, in joint work with Pilkiewicz (Pilkiewicz & Pottier, 2011), he proposes the use of *fates*, which enable reasoning about *monotonic* state in a manner rather similar to the state transition systems in our Kripke model. Second, in a brief unpublished note (Pottier, 2009), he sets forth a *generalized* version of the anti-frame rule that permits reasoning about well-bracketed state change.

While there are clear analogies between these extensions and our public/private state transitions, determining a precise formal correspondence is likely to be difficult because the methods are tailored to different purposes. On the one hand, Pottier's type systems are richer than that of ML, and thus his techniques can be used to verify correctness of some interesting programs that exploit the advanced features of his type systems. On the other hand, some equivalences—like our "deferred divergence" example from Section 5.2—do not seem to be easily expressible as "unary" typechecking problems and thus cannot seemingly be handled by Pottier's method. Moreover, like Sumii (2009), Pottier restricts attention to languages that support higher-order state but no control effects.

Pottier's anti-frame rule has only recently been proved sound, first in a relatively idealized setting (Schwinghammer *et al.*, 2010), and then, in Schwinghammer *et al.*, (2012), for the type-and-capability system (without regions) in which it was originally proposed (Pottier, 2008). The model in Schwinghammer *et al.*, (2012) also covers the generalized anti-frame rule, but not the extension with fates to reason about monotonic state mentioned above (Pilkiewicz & Pottier, 2011).

Other related work. Seminal work on operational reasoning about state and control was conducted by Felleisen & Hieb (1992) and Mason & Talcott (1991), but the proof principles they developed are relatively weak in comparison to the ones afforded by our model.

More recently, Yoshida *et al.* (2008) proposed a Hoare-style logic for reasoning about higher-order programs with local state, but it does not handle abstract types, nor does it permit the kind of reasoning achieved by our STSs. Dreyer *et al.* (2010) have devised a relational modal logic that accounts for the essential aspects of the ADR model. In the future, we hope to generalize that logic to account for the additional features we have proposed here. Lastly, Hur & Dreyer (2011) have developed logical relations for relating ML and assembly-language programs, based on the model we have presented here. They show how private and public transitions may be used to conveniently model low-level calling conventions concerning the stack and callee-save registers, as well as irreversible state changes in self-modifying code.

Acknowledgments

We would like to thank Chung-Kil Hur for suggesting that a restriction we had imposed on inconsistent states in an earlier draft of this paper was unnecessary. He was right, as usual.

References

Abramsky, S., Honda, K. & McCusker, G. (1998) A fully abstract game semantics for general references. *Proceedings of IEEE Symposium on Logic in Computer Science (LICS).*

Ahmed, A. (2004) *Semantics of Types for Mutable State.* PhD. thesis. Princeton University.

Ahmed, A., Dreyer, D. & Rossberg, A. (2009) State-dependent representation independence. *Proceedings of ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL).*

Appel, A. & McAllester, D. (2001) An indexed model of recursive types for foundational proof-carrying code. *ACM Trans. Program. Lang. Syst.* **23**(5), 657–683.

Benton, N. & Hur, C.-K. (2009) Biorthogonality, step-indexing and compiler correctness. *Proceedings of ACM SIGPLAN International Conference on Functional Programming (ICFP).*

Benton, N. & Tabareau, N. (2009) Compiling functional types to relational specifications for low level imperative code. *Proceedings of ACM SIGPLAN Workshop on Types in Language Design and Implementation (TLDI).*

Birkedal, L., Møgelberg, R., Schwinghammer, J. & Støvring, K. (January 2011) First steps in synthetic guarded domain theory: Step-indexing in the topos of trees. *Proceedings of IEEE Symposium on Logic in Computer Science (LICS).*

Bohr, N. (2007) *Advances in Reasoning Principles for Contextual Equivalence and Termination.* PhD. thesis. IT University of Copenhagen.

Dreyer, D., Ahmed, A. & Birkedal, L. (2011) Logical step-indexed logical relations. *Logical Methods Comput. Sci.* **7**(2:16), 1–37.

Dreyer, D., Neis, G. & Birkedal, L. (2012) *The Impact of Higher-Order State and Control Effects on Local Relational Reasoning (Technical Appendix).* Tech. Rep. MPI-SWS-2012-001. Max Planck Institute for Software Systems (MPI-SWS), Germany. Available at: http://www.mpi-sws.org/tr/2012-001.pdf.

Dreyer, D., Neis, G., Rossberg, A. & Birkedal, L. (2010) A relational modal logic for higher-order stateful ADTs. *Proceedings of ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL).*

Felleisen, M. & Hieb, R. (1992) The revised report on the syntactic theories of sequential control and state. *Theor. Comput. Sci.* **103**(2), 235–271.

Friedman, D. & Haynes, C. (1985) Constraining control. *Proceedings of ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL).*

Ghica, Dan R. & McCusker, G. (2000) Reasoning about Idealized Algol using regular languages. *Proceedings of International Colloquium on Automata, Languages and Programming (ICALP).*

Hur, C.-K. & Dreyer, D. (2011) A Kripke logical relation between ML and assembly. *Proceedings of ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL).*

Hur, C.-K., Dreyer, D., Neis, G. & Vafeiadis, V. (2012) The marriage of bisimulations and Kripke logical relations. *Proceedings of ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL).*

Johann, P. (2003) Short cut fusion is correct. *J. Funct. Program.* **13**(4), 797–814.

Johann, P., Simpson, A. & Voigtländer, J. (2010) A generic operational metatheory for algebraic effects. *Proceedings of IEEE Symposium on Logic in Computer Science (LICS).*

Johann, P. & Voigtländer, J. (2006) The impact of *seq* on free theorems-based program transformations. *Fundam. Inform.* **69**(1–2), 63–102.

Koutavas, V. & Lassen, S. (February 2008) *Fun with Fully Abstract Operational Game Semantics for General References.* Unpublished.

Koutavas, V. & Wand, M. (2006) Small bisimulations for reasoning about higher-order imperative programs. *Proceedings of ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL).*

Krivine, J.-L. (1994) Classical logic, storage operators and second-order lambda-calculus. *Ann. Pure Appl. Logic* **68**, 53–78.

Laird, J. (1997) Full abstraction for functional languages with control. *Proceedings of IEEE Symposium on Logic in Computer Science (LICS).*

Laird, J. (2007) A fully abstract trace semantics for general references. *Proceedings of International Colloquium on Automata, Languages and Programming (ICALP).*

Lassen, S. B. & Levy, P. B. (2007) Typed normal form bisimulation. *Proceedings of Conference on Computer Science Logic (CSL).*

Lassen, S. B. & Levy, P. B. (2008) Typed normal form bisimulation for parametric polymorphism. *Proceedings of IEEE Symposium on Logic in Computer Science (LICS).*

Mason, I. & Talcott, C. (1991) Equivalence in functional languages with effects. *J. Funct. Program.* **1**(3), 287–327.

Morris, J. H., Jr. (1968) *Lambda-Calculus Models of Programming Languages.* PhD. thesis. Massachusetts Institute of Technology.

Murawski, A. S. (2005) Functions with local state: Regularity and undecidability. *Theor. Comput. Sci.* **338**(1–3), 315–349.

Murawski, A. S. & Tzevelekos, N. (2011) Game semantics for good general references. *26th Annual IEEE Symposium on Logic in Computer Science.* Washington, DC, USA: IEEE Computer Society, pp. 75–84.

Murawski, A. S. & Walukiewicz, I. (2008) Third-order Idealized Algol with iteration is decidable. *Theor. Comput. Sci.* **390**(2–3), 214–229.

O'Hearn, P. & Reddy, U. (1995) Objects, interference, and the Yoneda embedding. *Proceedings of Conference on the Mathematical Foundations of Programming Semantics (MFPS).*

Pilkiewicz, A. & Pottier, F. (2011) The essence of monotonic state. *Proceedings of ACM SIGPLAN Workshop on Types in Language Design and Implementation (TLDI).*

Pitts, A. M. (1996) Reasoning about local variables with operationally-based logical relations. *Proceedings of IEEE Symposium on Logic in Computer Science (LICS).*

Pitts, A. (2005) Typed operational reasoning. In *Advanced Topics in Types and Programming Languages*, Pierce, B. C. (ed), Chap. 7. MIT Press.

Pitts, A. & Stark, I. (1998) Operational reasoning for functions with local state. *Proceedings of International Workshop on Higher Order Operational Techniques in Semantics (HOOTS).*

Pottier, F. (2008) Hiding local state in direct style: A higher-order anti-frame rule. *Proceedings of IEEE Symposium on Logic in Computer Science (LICS).*

Pottier, F. (2009) *Generalizing the Higher-Order Frame and Anti-Frame Rules.* Unpublished.

Reddy, U. S. & Dunphy, B. P. (2011) An automata-theoretic model of objects. *Proceedings of International Workshop on Foundations of Object-Oriented Languages (FOOL).*

Sangiorgi, D., Kobayashi, N. & Sumii, E. (2011) Environmental bisimulations for higher-order languages. *ACM Trans. Program. Lang. Syst.* **33**(1:5), 1–69.

Schwinghammer, J., Birkedal, L., Pottier, F., Reus, B., Støvring, K. & Yang, H. (2012) A step-indexed Kripke model of hidden state. In *Mathematical Structures in Computer Science.* To appear.

Schwinghammer, J., Yang, H., Birkedal, L., Pottier, F. & Reus, B. (2010) A semantic foundation for hidden state. *Proceedings of Foundations of Software Science and Computation Structures (FOSSACS).*

Støvring, K. & Lassen, S. B. (2007) A complete, co-inductive syntactic theory of sequential control and state. *Proceedings of ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL).*

Sumii, E. (2009) A complete characterization of observational equivalence in polymorphic λ-calculus with general references. *Proceedings of Conference on Computer Science Logic (CSL).*

Sumii, E. & Pierce, B. (2007) A bisimulation for type abstraction and recursion. *J. ACM* **54**(5), 1–43.

Thielecke, H. (2000) On exceptions versus continuations in the presence of state. *Proceedings of European Symposium on Programming (ESOP).*

Yoshida, N., Honda, K. & Berger, M. (2008) Logical reasoning for higher-order functions with local state. *Logical Methods Comput. Sci.* **4**(4:2), 1–68.